MCR-86-675

VERSION 1

PROGRAM MAINTENANCE MANUAL

FOR

NICKEL CADMIUM BATTERY EXPERT SYSTEM

NAS8-35922

DEVELOPED FOR

NASA/MARSHALL SPACE FLIGHT CENTER

HUNTSVILLE, ALABAMA

PREPARED BY

MARTIN MARIETTA CORPORATION

AEROSPACE DIVISION

DENVER, COLORADO

OCTOBER 1986

TABLE OF CONTENTS

TABLE OF CONTENTS CONTINUED

ILLUSTRATIONS

SECTION 1. GENERAL

1.1 Purpose of the Program Maintenance Manual.

The objective for writing this Program Maintenance Manual for project Nickel-Cadmium Battery Expert System (NICBES), Contract Number : NAS8-35922, is to provide the maintenance programmer personnel with the information necessary to effectively maintain or enhance the system.

1.2 Project References.

NICBES is an Expert System for fault diagnosis and advice of the Nickel-Cadmium Batteries found in the Hubble Space Telescope (HST) Electrical Power System (EPS) Testbed located at Marshall Space Flight Center (MSFC). NICBES resides on a dedicated IBM-PC AT and operates in two modes. The first mode is the Data-Handler which is written in MICROSOFT C. The second mode is the Expert System which is written in ARITY PROLOG, a logical programming language. The following documents and manuals serve as reference materials for NICBES:

- NICBES User's Manual - September 1986
- IBM-PC AT Manuals
- ARITY PROLOG Manuals - Version 4.1
- MICROSOFT C Manuals

1.3 Terms and Abbreviations.

AHI	- ampere hours in
AHO	- ampere hours out
bprc	- battery protection and reconditioning circuits
CCC	- charge current controllers
DOD	- depth of discharge
EOC	- end of charge
EOD	- end of discharge
EOF	- end of file
EPS	- electrical power system
HST	- Hubble Space Telescope
MMDA	- Martin Marietta Denver Aerospace
MSFC	- Marshall Space Flight Center
NICBES	- Nickel-Cadmium Battery Expert System
SOW	- Statement of Work
SPA	- solar panel array

SECTION 2. SYSTEM DESCRIPTION

2.1 System Application.

NICBES was developed as an assistant for engineers working on the HST EPS Testbed to aid in decision making with regard to the Nickel-Cadmium Batteries. NICBES analysis depends on the particular testbed configuration at MSFC, see Figure 1, and the particular Battery manufacturer.

2.2 Security.

There are no security requirements.

2.3 General Description.

NICBES, as programmed for the IBM-PC AT, a single tasking computer, requires two independent processes. The first is the Data-Handler which processes incoming telemetry every one minute. Input to the Data-Handler comes from the DEC LSI-11 over a RS232 to the IBM-PC AT. Each telemetry burst contains 370 integer and floating point values preceeded by the character 'A'. There are 96 minutes in one orbit. An orbit is composed of a discharge and charge phase. An orbit starts at the beginning of the discharge phase and ends at the completion of the charge phase. Once this process is completed (for 12 orbits total), the Expert System of NICBES can be run. Input to the Expert System consists of processed data files output by the Data-Handler. Output from the Expert System includes fault diagnosis, battery status and advice, plus decision support. Any Expert System screen displays can be routed to the STAR-SD-15 Printer for hardcopy.

2.4 Program Description.

Program Descriptions for NICBES will be given in two sets. First the Data-Handler will be described and then the Expert System.

2.4.1 Data-Handler.

The Data-Handler, written in MICROSOFT C, is installed on the IBM-PC AT according to the procedures listed in the MICROSOFT C Manual. All the following programs can be found under C:\USR . This is also where they should be executed. The data output files are also written to this directory. See Figure 1 for Data-Handler Flow Diagram.

2.4.1.1 datacl.bat

datacl.bat creates the executable for the Data-Handler. It compiles data_hdl.c, the main control routine, and then links all the other object files needed. The result is data_hdl.exe which invokes the Data-Handler. The user simply types 'data_hdl' from the DOS prompt to execute the Data-Handler. If changes are made to any of the 'C' programs, they must be recompiled ('msc filename.c;') and then relinked ('datacl') to create a new executable. 'printf' statements used in development have been left in the programs but commented out. These statements are can be reinstated for debugging purposes.

DATA-HANDLER FLOW DIAGRAM

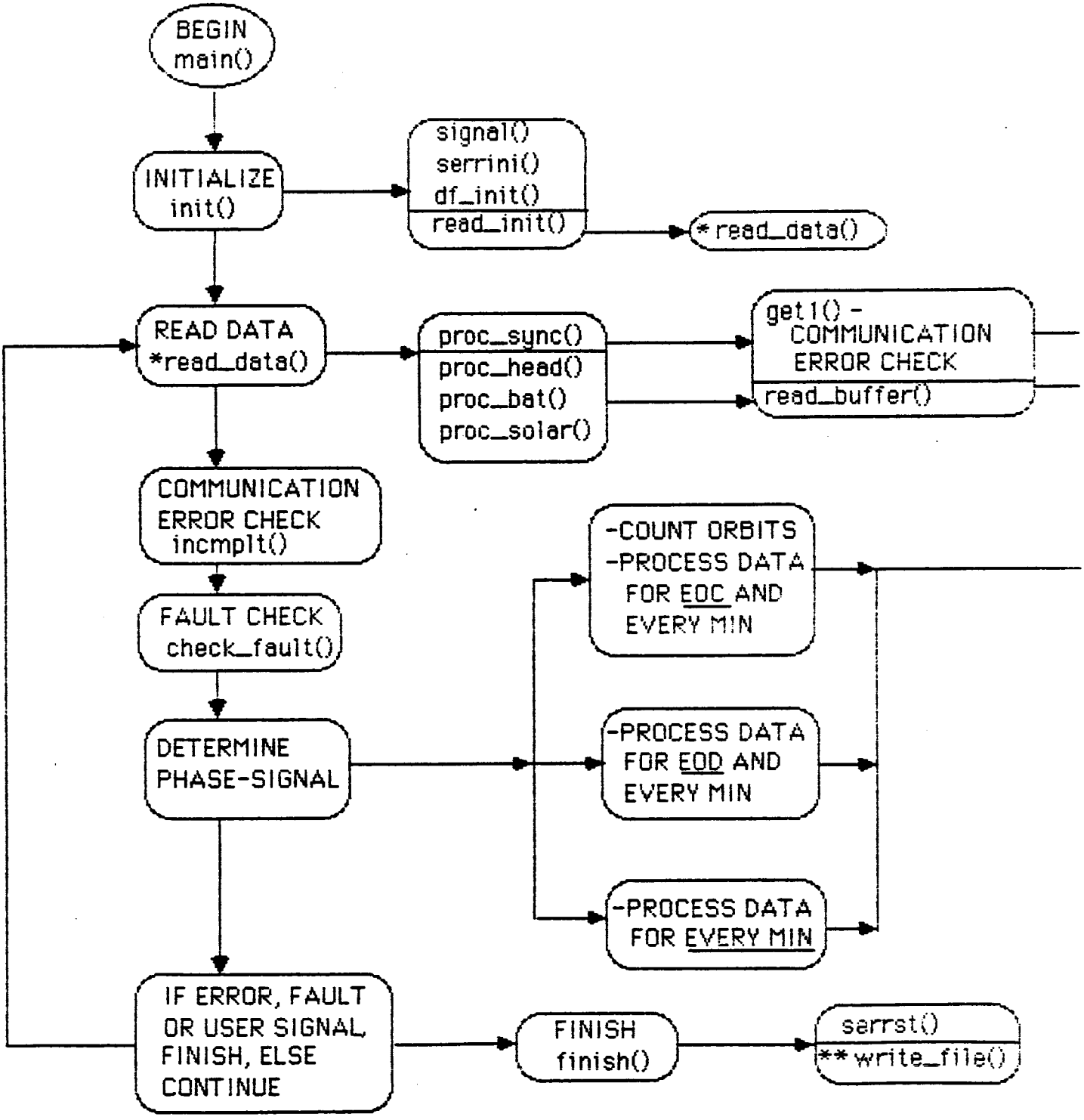
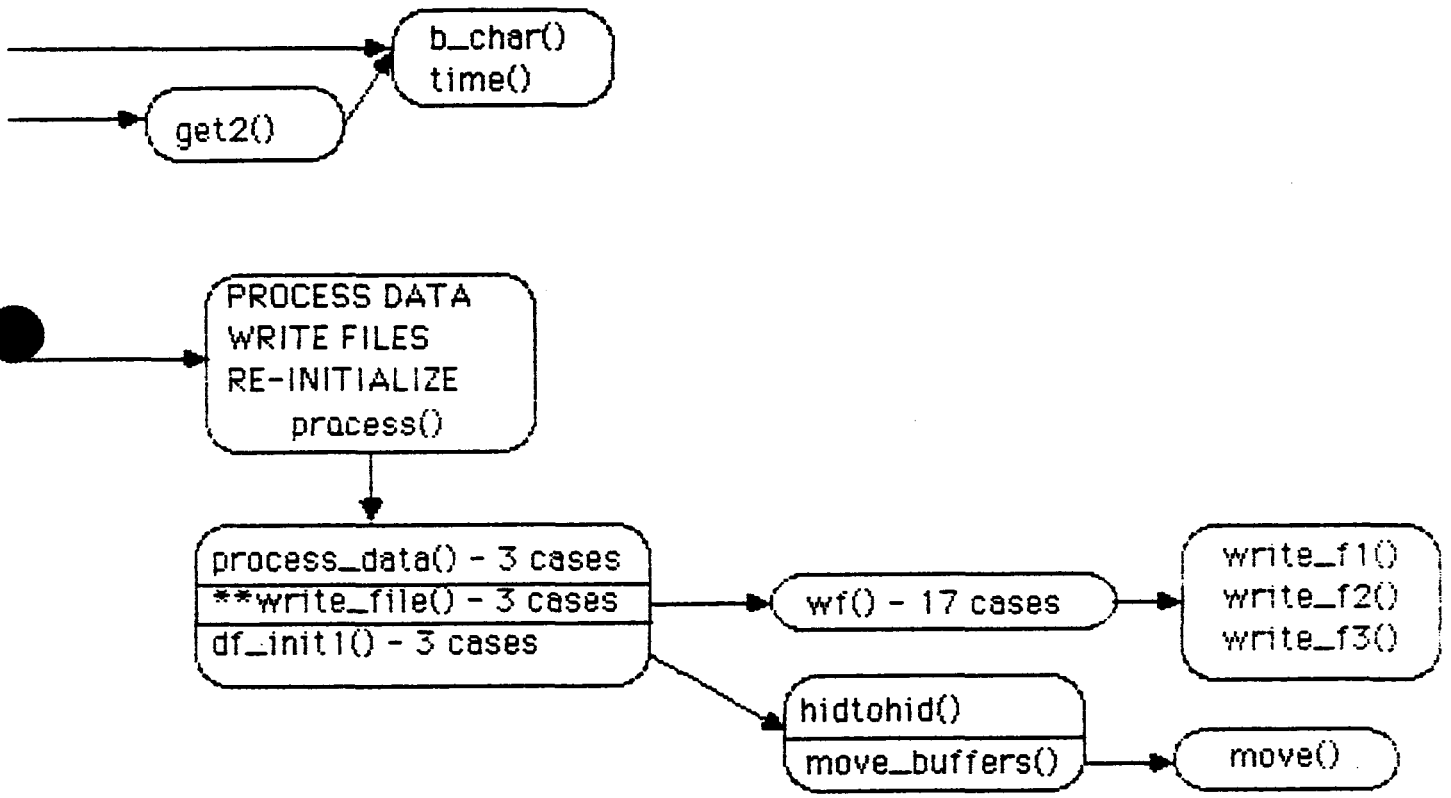


FIGURE 1


b_char()
time()
get2()
PROCESS DATA
WRITE FILES
RE-INITIALIZE
process()
process_data() - 3 cases
**write_file() - 3 cases
df_init1() - 3 cases
wf() - 17 cases
write_f1()
write_f2()
write_f3()
hidtohid()
move_buffers()
move()

2.4.1.2 hst.h

hst.h contains all the header items for the Data-Handler. It starts with define statements for errors, phases and data files. It also lists the needed system include files. Last are the structure and matrix definitions for global arrays.

Define Statements:

```
    SUCCESS          1
    FAIL             0
    EOC              2
    EOD              3
    EVMIN            1
    DEFLT       -9999.0
    SHOWF(N)DAT      N  where N = 1 to 13
    CURF(N)DAT       N  where N = 14 to 16
    FAULTDAT        17
```

Include files:

stdio.h, stdlib.h, process.h and errno.h

Global Structure for storing telemetry run, hid[0] = last run, hid[1] = current run.

hid[i].year	Int	i = 0 to 1
hid[i].day	Int	col 0 = last telemetry run
hid[i].hour	Int	col 1 = current telemetry run
hid[i].min	Int	
hid[i].sec	Int	
hid[i].orbit	Int	
hid[i].phase	Int	0 if discharge, 1 if charge
hid[i].day_min	Int	minute in charge phase
hid[i].night_min	Int	minute in discharge phase
hid[i].batd[j].battno	Int	j = 0 to 5, for 6 batteries
hid[i].batd[j].cellv[k]	Real	k = 0 to 22, for 23 cells
hid[i].batd[j].cellp[k]	Real	per battery
hid[i].batd[j].batv	Real	p => pressure, v => volts,
hid[i].batd[j].batc	Real	c => current
hid[i].batd[j].bprcc	Real	batt reconditioning current
hid[i].batd[j].batemp[k]	Real	6 temp sensors per battery
hid[i].batd[j].batrecond	Int	int flag for reconditioning
hid[i].spac[k]	Real	k = 0 to 12, for 13 SPAs
hid[i].bd[k].busv	Real	k = 0 to 2, for 3 busses
hid[i].bd[k].busc		

The global matrices and arrays which support the data files are documented in Appendix A of the code listing for hst.h

2.4.1.3 data_hdl.c

data_hdl.c is the main driver for the Data-Handler. Following is a description of all the 'C' routines in this program (main(), init(), read_init(), finish(), sig_catch(), process() and incomplete()).

Identification: main

Function: main is the Data-Handler driver. It controls the program flow and determines the times for events to happen.

Input: No Input, although argc and argv make it possible to easily add inputs.

Processing: Keeps track of Orbit and Phase. Controls program flow by its routine calling sequence.

Output: Error Message written to screen if more than CILIMIT consecutive incomplete telemetry runs.
Error Message if fault is detected.
Message written to screen at the completion of each orbit.

Local Variables:
- orbitno - Counter for number of orbits
- err - recieves return from read_data()
- phase_signal - EOC, EOD or EVMIN
- phase1 - phase of previous telemetry run
- phase2 - phase of current telemetry run

Global Variables:
- EOC, EOD, EVMIN
- FAULTDAT
- fault[]
- FAIL
- hid[]

Interfaces: Calls init(), process(), finish(), sig_catch(), incomplete()
- read_dat.c - read_data()
- writ_fil.c - wf().
- process.c - check_fault()

Error Handling: If read_data encounterd an EOF while reading the telemetry stream, FAIL is returned. If there are CILIMIT consecutive incomplete runs, Data-Handler is shutdown after the fault flag is set to 1.
If a fault was detected the fault flag is set to 1 and the Data-Handler is exited.

Identification:. init

Function: Sets the communication port, initializes interrupt handler, and calls buffer initialization and read telemetry initialization routines.

Input: No input.

Processing: No processing.

Outputs: Error message stating that an EOF was found while reading the telemetry stream and that the system is shutting down.

Local Variables:
err - receives return value from read_init().
port - communication port

Global Variables: None

Interfaces: Called by main.
Calls system signal() and set_port(), read_init(), interrupt handler - serini() and process.c - df_init().

Error Handling: If read_init() returns FAIL, system will shutdown.

Identification: read_init

Function: Initializes telemetry reading to first full orbit. An orbit starts at the beginning of the discharge phase.

Input: No input.

Processing: Reads the telemetry stream until night_min = 1, start of discharge phase. This signifies the start of an orbit. process() is then called to process this first set of data.

Output: Prints message to the screen "Starting first full orbit".

Local Variables: err - receives return value from read_data().

Global Variables:
FAIL, DEFLT, EVMIN
hid[]

Interfaces: Called by init().
Calls process(), incomplete(), read_dat.c - read_data().

Error Handling: If read_data() returns FAIL, control is passed back to init() also with a FAIL message.

Identification: finish

Function: Exits Data Handler.

Input: No input.

Processing: Ends interrupt handler, writes data output files and calls exit().

Output: No output.

Local Variables: None.

Global Variables: EOC, EOD

Interfaces: Called by main(), init(), sig_catch(), incomplete() or read_dat.c - get2().
Calls writ fil.c - write_file(), system exit()
and interrupt handler - serrst().

Error Handling: None.

Identification: sig_catch

Function: Catches '^C' signal input by operator to halt Data-Handler.

Input: User signal input '^C'.

Processing: No processing.

Output: Writes message to screen, "Interrupt caught; exiting!"

Local Variables: None.

Global Variables: None

Interfaces: Calls finish().

Error Handling: None.

Identification: process

Function: Calls the routines necessary to process the telemetry.

Input: phase_signal - EOC, EOD or EVMIN
orbitno - 0 to N, N is the number of orbits.

Processing: No processing.

Output: No output.

Local Variables: None.

Global Variables: None.

Interfaces: Calls process.c - process_data(), df_initl(),
writ_fil.c - write_file().

Error Handling: None.

Identification: incomplete()

Function: Checks for CILIMIT number of consecutive incomplete telemetry bursts.

Input: None.

Processing: If there are CILIMIT consecutive incomplete telemetry runs fault flags are set, fault.dat is written and finish() is called.

Output: Error message.

Local Variables: None.

Global Variables:
CILIMIT - Limit for consecutive incomplete telemetry runs
conseq_incmplt - counter for incomplete telemetry bursts
FAULTDAT
fault[]

Interfaces: Called by main() and read_init().
Calls finish() and writ_fil.c - wf().

2.4.1.4 read dat.c

read__dat.c is the routine which reads the telemetry from the interrupt handler buffer. Following is a description of all the 'C' routines in this file (read__data(), proc__sync(), proc__head(), proc_bat(), proc solar(), read_buffer(), get1() and get2()).

Identification: read_data

Function: Read telemetry from DEC LSI-11 over RS232 every 1 minute.

Input: No input, however telemetry from HST EPS Testbed, 370 values preceded by 'A', is utilized.

Processing: Calls routines to read telemetry.

Output: Returns FAIL or SUCCESS.
Telemetry is placed in structured arrary hid[1] for later processing. A description of the structure of hid follows:

Local Variables err - return value from proc_head(), proc_bat(), proc_solar().

Global Variables:
T1LIMIT, T2LIMIT
buf[] - intermediate character storage array.
count - keeps count of data values per telemetry burst.
FAIL, SUCCESS

Interfaces: Called by main().
Calls proc_sync(), proc_head(), proc_bat(), proc_solar().

Error Handling: If proc_head(), proc_bat() or proc_solar() return a FAIL, reading is stopped and control is returned to the read_data() call in main().

Identification: proc_sync

Function: Synchronize reading data to start of telemetry burst.

Input: No input, but uses telemetry burst, first character is 'A'.

Processing: Checks input characters until character 'A" is found.
Also checks for a shutdown signal from the DEC LSI-11.

Output: Can write message to screen "Sync Received!"
Can write message to screen "Received shutdown signal from DEC LSI-11!"

Local Variables:
cc - character found in the interrupt handler's buffer.

Global Variables:
fault[]
FAULTDAT

Interfaces: Called by read_data().
Calls data_hdl.c - finish(), writ_fil.c - wf(), get1() and get2().

Error Handling: Error Message if shutdown signal from DEC LSI-11.

Identification: proc_head

Function: Read header data from input buffer.

Input: No input, but uses telemetry stream.

Processing: Read header data from input buffer into buf[]. Use sscanf to put characters into integer format.

Output: Integer header data put into global structured array hid[1].
Returns FAIL or SUCCESS.

Local Variables err - return value of read_buffer().

Global Variables:
FAIL, SUCCESS
hid[]
buf[]

Interfaces: Called by read_data().
Calls read_buffer().

Error Handling: If read_buffer() returns a FAIL, proc_head() stops and returns FAIL to read_data().

Identification: proc_bat

Function: Read battery data from input buffer.

Input: No input, but uses telemetry, 57 values for each of the 6 batteries.

Processing: Read battery data from input buffer into buf[]. Use sscanf to put characters into integer and floating point format.

Output: Integer and real battery data put into global structured array hid[1]. Returns FAIL or SUCCESS.

Local Variables err - return value from read_buffer().

Global Variables:
FAIL, SUCCESS
hid[]
buf[]

Interfaces: Called by read_data().
Calls read_buffer().

Error Handling: If read_buffer() returns a FAIL, proc_bat() stops and returns FAIL to read_data().

Identification: proc_solar

Function: Read SPA and bus data from input buffer.

Input: No input, but uses telemetry, 13 SPA and 6 bus values.

Processing: Read SPA and bus data from input buffer into buf[]. Use sscanf to put characters into floating point format.

Output: SPA and bus data put into global structured array hid[1]. Returns FAIL or SUCCESS.

Local Variables err - return value from read_buffer().

Global Variables:
FAIL, SUCCESS
hid[]
buf[]

Interfaces: Called by read_data().
Calls read_buffer().

Error Handling: If read_buffer() returns a FAIL, proc_solar() stops and returns FAIL to read_data().

Identification: read_buffer

Function: Puts characters from interrupt handler buffer into buf[].

Input: k is the actual number of telemetry values to be read.

Processing: Get next character in interrupt handler buffer, this includes newlines and <CR>s. Characters are put in buf[].

Output: Returns found characters in buf[] to calling routine.
Returns FAIL or SUCCESS.

Local Variables nl_count - counts telemetry values read by counting newlines.

Global Variables:
count
buf[]
EOF
FAIL, SUCCESS

Interfaces: Called by proc_head(), proc_bat(), proc_solar().
Calls getl().

Error Handling: If EOF is encountered while reading the telemetry stream, read_buffer() is stopped and FAIL is returned to the calling routine.

Identification getl()

Function: Reads characters from interrupt handler buffer during telemetry burst.

Input: None.

Processing: Gets characters from interrupt handler buffer. Checks time and count to insure complete telemetry runs are read.

Output: Returns EOF or character.

Local Variables:
cc - character read from interrrupt handler buffer.
t0 - start time for timer.
tn - current time
timer - difference between tn and t0.

Global Variables:
count
EOF
TILIMIT - maximum time to wait for next character

Interfaces: Called by read_buffer() and proc_sync().
Calls interrupt handler b_char().

Error Handling: If a character is not read within a given time limit it is assumed to be an incomplete telemetry burst. Passes EOF back to read_buffer().

Identification get2()

Function: Reads characters from interrupt handler buffer at the start of a telemetry run.

Input: . None.

Processing: Gets characters from interrupt handler buffer. Checks time to insure the beginning of telemetry is read within T2LIMIT time limit.

Output: Returns character. Writes Error Message if time limit exceeded.

Local Variables:
cc - character read from interrrupt handler buffer.
t0 - start time for timer.
tn - current time
timer - difference between tn and t0.

Global Variables:
EOF
T2LIMIT - maximum time to wait for next character
fault[]
FAULTDAT

Interfaces: Called by proc_sync().
Calls interrupt handler b_char().

Error Handling: If a character is not read within a given time limit it is assumed that a telemetry run is missed.
Sets fault flag to 1, writes fault.dat, writes Error Message "No communication in 3 minutes; exiting!", and exits.

2.4.1.5 process.c

process.c processes the current telemetry which has been stored in hid[] by read_dat.c. Following is a description of all the 'C' routines in this file (df_init(), df_initl(), hidtohid(), move_buffers(), move(), process_data(), check_fault()).

Identification: df_init

Function: Start up initialization counters and globally defined structures, matrices and arrays.

Input: No input, but uses globally defined structures and arrays.

Processing: Sets counters, matrices and arrays to 0 or DEFLT.

Output: No output.

Local Variables: None.

Global Variables:

DEFLT - default value = -9999.0. Signifies missing data.
no_druns - number of discharge runs in an orbit.
no_cruns - number of charge runs in an orbit.
fault[]
FAULTDAT

DATA FILE	CORRESPONDING BUFFER
showf1.dat	eod_voltage(6,12)
showf2.dat	hc_voltage(6,12)
	high_buffer(6)
showf3.dat	rc_ratio(6,12)
	ahi(6)
	ahoo(6)
showf4.dat	cv_eod_hv(6,12)
	cv_eod_lv(6,12)
	cv_eod_av(6,12)
showf5.dat	cv_hc_hv(6,12)
	cv_hc_lv(6,12)
	cv_hc_av(6,12)
showf6.dat	cellv_eod(6,12)
showf7.dat	cellv_hc(6,23)
showf8.dat	avgt(6,48)
showf9.dat	avg_temp(6,12)
	avg_temp_buffer(6)
showf10.dat	cp_eod(6,23)
	cp_eoc(6,23)
showf11.dat	time_tc(6,12)
	trickle(6)
showf12.dat	rc_orbit(6)
	bc_drc(6,48)
showf13.dat	aho(6,12)
curf2.dat	batt_avg(6)

Interfaces: Called by data_hdl.c - init().

Error Handling: None.

Identification: df_initl

Function: After each telemetry burst is read, certain counters, structures and arrays used for processing need to be re-initialized. There are 3 cases - EOC, EOD or EVMIN.

Input: phase_signal - EOC, EOD or EVMIN. Also uses globally defined structures, arrays and matrices.

Processing: Sets counters, variables and arrays to 0 or DEFLT (-9999.).

Output: No output.

Local Variables: None.

Global Variables:

DEFLT - default value = -9999.0. Signifies missing data.
no_druns - number of discharge runs in an orbit.
no_cruns - number of charge runs in an orbit.
EOC, EOD, EVMIN

DATA FILE	CORRESPONDING BUFFER
showf2.dat	high_buffer(6)
showf3.dat	ahoo(6)
	ahi(6)
showf6.dat	cellv_eod(6,12)
showf7.dat	cellv_hc(6,23)
showf8.dat	avgt(6,48)
showf9.dat	avg_temp_buffer(6)
showf10.dat	cp_eod(6,23)
	cp_eoc(6,23)
curf2.dat	batt_avg(6)

Interfaces: Called by data_hdl.c - process.

Error Handling: None.

Identification: hidtohid

Function: After each telemetry burst is read and processed, the data in hid[1] is put in hid[0] to prepare for next data burst which will be stored in hid[1].

Input: No input.

Processing: Puts hid[1] column into hid[0] column.

Output: No output.

Local Variables: None.

Global Variables:
hid[]
DEFLT

Interfaces: Called by df_initl().

Error Handling: None.

Identification: move_buffers

Function: Prepares the globally defined matrices for move().
The affected matrices are those associated with data files containing data for 12 orbits. When more than 12 orbits have been processed, the arrays need to be shifted so that they contain the only the last 12 orbit's data.
There are two cases - EOC and EOD.

Input: phase_signal - tells whether charge or discharge phase.

Processing: Prepares matrices and then calls move().

Output: No output.

Local Variables None.

Global Variables:
EOC, EOD

DATA FILE	CORRESPONDING BUFFER
showf1.dat	eod_voltage(6,12)
showf2.dat	hc_voltage(6,12)
showf3.dat	rc_ratio(6,12)
showf4.dat	cv_eod_hv(6,12)
	cv_eod_lv(6,12)
	cv_eod_av(6,12)
showf5.dat	cv_hc_hv(6,12)
	cv_hc_lv(6,12)
	cv_hc_av(6,12)
showf9.dat	avg_temp(6,12)
showf11.dat	time_tc(6,12)
showf13.dat	aho(6,12)

Interfaces: Called by process_data().
Calls move().

Error Handling: None.

Identification: move

Function: When the Data-Handler continues after 12 completed orbits, the 6x12 matrices must loose their first column, the remaining data must be shifted one column to the left and the next orbit's data will be put in the 12th column. You hence always have the latest 12 orbits.

Input: buffer[] to be moved.

Processing: Shift columns in buffer one column to the left, dropping the first column. Set the 12th column to DEFLT (-9999.).

Output: No output.

Local Variables: None.

Global Variables: DEFLT - default value = -9999.0.

Interfaces: Called by move_buffers().

Error Handling: None.

Identification: process_data

Function: 96 minutes of telemetry making up each orbit, are summarized mathematically in preparation for writing the data to output files. There are 3 cases - EOC, EOD and EVMIN.

Input: phase_signal - EOC, EOC or EVMIN.
orbitno - 0 to N, where N is the number of orbits.

Processing: Data is prepared for showf(n).dat, n = 1 to 13 and curf(n), n = 1 to 3. Following, in Section 3.3.2 on data bases, a description is given of each data file's functional requirements.

Output: No output.

Local Variables:
col - 0 to 11, matches orbit to column number of matrices.
sum - variable used to sum 6 temperature sensors per battery.
avg - sum / 6 to give average temperature of battery per min.
current_min - sum of night_min and day_min.
jj - flag to do processing on even minutes.
x1 - used to find maximums.
x2 - used to find minumums.
x3 - used to find averages.
x - miscellaneous variable.

Global Variables:

DEFLT - default value = -9999.0. Signifies missing data.
no_druns - number of discharge runs in an orbit.
no_cruns - number of charge runs in an orbit.
DCHGLIMIT - necessary number of discharge runs per orbit.
CHGLIMIT - necessary number of charge runs per orbit.

DATA FILE	CORRESPONDING BUFFER
showf1.dat	eod_voltage(6,12)
showf2.dat	hc_voltage(6,12)
	high_buffer(6)
showf3.dat	rc_ratio(6,12)
	ahi(6)
	ahoo(6)
showf4.dat	cv_eod_hv(6,12)
	cv_eod_lv(6,12)
	cv_eod_av(6,12)
showf5.dat	cv_hc_hv(6,12)
	cv_hc_lv(6,12)
	cv_hc_av(6,12)
showf6.dat	cellv_eod(6,12)
showf7.dat	cellv_hc(6,23)
showf8.dat	avgt(6,48)
showf9.dat	avg_temp(6,12)
	avg_temp_buffer(6)
showf10.dat	cp_eod(6,23)
	cp_eoc(6,23)
showf11.dat	time_tc(6,12)
	trickle(6)
showf12.dat	rc_orbit(6)
	bc_drc(6,48)
showf13.dat	aho(6,12)
curf2.dat	batt_avg(6)

The indented buffers are working arrays which support the main arrays. The data files followed by blanks require no processing but directly use telemetry from hid[1].
In addition process_data determines if it is necessary to call move_buffers and keeps count of number of charge and discharge runs per orbit.

Interfaces: Called by main().

Error Handling: Checks that no division by zero occurs.
Checks that enough charge and discharge runs have been recorded to validate the processed data. Else DEFLT is left in the global matrices and process() stops.

Identification: check_fault()

Function: Detects faults in telemetry.

Input: No Input.

Processing: There are four fault categories that are checked:

1. Power Supplies
   a. SPA current < 5 amps during first 5 minutes of charge phase.
   b. SPA current >= 8 amps for 1-SPAs (1,3,5,7,9,11).
      SPA current >= 16 amps for 2-SPAs (2,4,6,8,10,12,13).
   c. SPA current > 5 amps during discharge phase.

2. Batteries
   a. Cell voltage <= 0 volts for any cell in any battery.
   b. Cell voltage > 1.55 volts for any cell in any battery.

3. Load Banks
   a. Sum of 3 bus currents > 99 amps.
   b. Load < 5 amps on any single bus during discharge phase.

4. Temperature
   a. Average of the 6 temperature sensors > 25 C or < -10 C.

Output: Returns FAIL or SUCCESS

Local Variables:
x - miscellaneous floating point number.
sum - sum of various arrays.

Global Variables:
FAIL, SUCCESS
hid[]

Interfaces: Called by data_hdl.c - main().

Error Handling: Returns FAIL if fault found, SUCCESS if not.

2.4.1.6 writ_fil.c

writ__fil.c contains the 'C' routines needed to write the processed data buffers to output files. Following is a description of all the 'C' routines in this file (wf(), write__f1(), write_f2(),write_f3(), write file()).

Identification: write_file

Function: Determines which data files should be written. There are 3 cases - EOC, EOD or EVMIN.

Input: phase_signal - EOC, EOD or EVMIN.

Processing: According to the time, wf is called to write the data output files. Following is a time chart:

TIME	DATA FILES
EOC	showf2.dat
	showf3.dat
	showf5.dat
	showf6.dat
	showf7.dat
	showf8.dat
	showf9.dat
	showf11.dat
	showf12.dat
EOD	showf1.dat
	showf4.dat
	showf6.dat
	showf10.dat
	statf1.dat
EVMIN	curf1.dat
	curf2.dat
	curf3.dat

fault.dat is written initially with fault flag = 0, and then only after fault flag is set to 1.

Output: Error message is written to the screen "Couldn't open 'filename'!".

Local Variables: err - return value from wf().

Global Variables:

EOC, EOD, EVMIN
SHOWF(N)DAT for N = 1 to 13
CURF(N)DAT for N = 1 to 3

Interfaces: Called by data_hdl.c - process() and finish().
Calls wf().

Error Handling: If a data file can not be opened, a message is written to the screen. No other action is taken.

Identification: wf

Function: Write output files for Expert System from globally defined matrices and arrays containing summarized telemetry. There is a case statement for each data file.

Input: Name of the data file to be written.

Processing: Open output files, write processed data from matrices in list format, then close output file.

Output: Data files to be used by the Expert System.
Returns FAIL or SUCCESS.

Local Variables:
err - return value from write_f1(), write_f2(), write_f3()
sfp - output file pointer.

Global Variables:
SHOWF(N)DAT for N = 1 to 13
CURF(N)DAT for N = 1 to 3, and FAULTDAT
FAIL, SUCCESS
hid[]

DATA FILE	CORRESPONDING BUFFER
showf1.dat	eod_voltage(6,12)
showf2.dat	hc_voltage(6,12)
showf3.dat	rc_ratio(6,12)
showf4.dat	cv_eod_hv(6,12)
	cv_eod_lv(6,12)
	cv_eod_av(6,12)
showf5.dat	cv_hc_hv(6,12)
	cv_hc_lv(6,12)
	cv_hc_av(6,12)
showf6.dat	cellv_eod(6,12)
showf7.dat	cellv_hc(6,23)
showf8.dat	avgt(6,48)
showf9.dat	avg_temp(6,12)
showf10.dat	cp_eod(6,23)
	cp_eoc(6,23)
showf11.dat	time_tc(6,12)
showf12.dat	rc_orbit(6)
	bc_drc(6,48)
showf13.dat	aho(6,12)
curf2.dat	batt_avg(6)

Interfaces: Called by write_file(),
data_hdl.c - main(), incomplete(),
read_dat.c - get2()..
Calls write_f1(), write_f2(), write_f3(),
system fopen(), fclose() and fprintf().

Error Handling: Returns FAIL if data file can not be opened.

Note: List format means that data is contained in brackets and separated by commas. A main list holds all the sublists and ends with a peroid. Following is an example for showf1.dat:

```
show(1,[[a1,b1,c1,d1,e1,f1,g1,h1,i1,j1,k1,l1],
    [a2,b2,c2,d2,e2,f2,g2,.............],
    [a3,.............................],
    [a4,.............................],
    [a5,.............................],
    [a6,.............................]]).
```

In this example there are 12 columns, one per orbit. Orbital data is listed in chronological order.

Identification: write_f1

Function: Write output files for Expert System from globally defined matrices and arrays containing summarized telemetry.

Input: filename, buffer and file number to be written to output file.

Processing: Open output files, write processed data in list format, then close output file.

Output: Data files to be used by the Expert System.
Returns FAIL or SUCCESS.

Local Variables file - output file pointer.

Global Variables: FAIL, SUCCESS

Interfaces: Called by wf() for showf1, showf2, showf3, showf9, statl.
Calls system fopen(), fclose() and fprintf().

Error Handling: Returns FAIL if data file can not be opened.

Identification: write_f2

Function: Write output files for Expert System from globally defined matrices and arrays containing summarized telemetry.

Input: filename, buffer and file number to be written to output file.

Processing: Open output files, write processed data in list format, then close output file.

Output: Data files to be used by the Expert System.
Returns FAIL or SUCCESS.

Local Variables: file - output file pointer.

Global Variables: FAIL, SUCCESS

Interfaces: Called by wf() for showf4 and showf5.
Calls system fopen(), fclose() and fprintf().

Error Handling: Returns FAIL if data file can not be opened.

Identification: write_f3

Function: Write output files for Expert System from globally defined matrices and arrays containing summarized telemetry.

Input: filename, buffer and file number to be written to output file.

Processing: Open output files, write processed data in list format, then close output file.

Output: Data files to be used by the Expert System.
Returns FAIL or SUCCESS.

Local Variables: file - output file pointer.

Global Variables: FAIL, SUCCESS

Interfaces: Called by wf() for showf6 and showf7.
Calls system fopen(), fclose() and fprintf().

Error Handling: Returns FAIL if data file can not be opened.

2.4.1.7 Interrupt Handler

The Interrupt Handler takes over control of the IBM poling technique for receiving data over a communication net. Instead each incoming character is retrieved in a buffer which can be accessed by the Data-Handler programs. This is to insure that the telemetry is read accurately and not written over.

These are the programs needed:

- serial.c and serial.obj
- com_cfns.c and com_cfns.obj
- com_fns.asm and com_fns.obj
- serset.asm and serset.obj
- fixup.asm and fixup.obj

Functions directly called from the Data-Handler are:

- data_hdl.c - init() calls serini()
  - finish() calls serrst()
- read_dat.c - get1() calls b_char()
  - get2() calls b_char()

Include files required are:

- serial.h
- entry.h
- asment.h
- asmexit.h
- sasment.h
- sasmexit.h

2.4.2 Expert System.

The Expert System is written in ARITY PROLOG which is installed on the IBM-PC AT according to the procedures listed in the ARITY PROLOG Manual. All the following programs can be found under C:\PROLOG. This is also where they should be executed. The set of data files to be used for the Expert System analysis need to be copied to C:\PROLOG. Unlike the 'C' programs, the PROLOG programs of made up of many predicates and control is implemented by predicate calls to other predicates. In a way this is similar to subroutines at a smaller level. The programs have been grouped to be modular. See Figure 2 for Expert System Flow Diagram.

2.4.2.1 start.prg

Identification: start

Function: Main control routine for the Expert System. It calls other segments of the Expert System via the utilization of user menus.

Input: fault.dat, curfl.dat and user responses to menus.

Processing: First fault.dat is checked to see if the fault flag has been set to 1. If so, faultd.prg is called to perform fault diagnosis. The user can then opt for more information. In this case or if no fault, the Main Menu is written to the screen from which the user can select from Plots and Graphs, Battery Status or Advice. Next the user is asked to select Battery.
Control is passed to one of the above 3 choices with the selected Battery. Further menus are shown for Plots and Graphs and for Advice. The user can always opt for another Battery selection or to Quit to the Main Menu where they can opt to Quit NICBES.

Output: Menus and control parameters which are passed to other portions of the Expert System telling what the user's choices are in response to menus, and which Battery to view.

Interfaces: Invoked by prolog.ini, the PROLOG initiation program. Start calls functions in faultd.prg, status.prg, advice.prg, showpak.prg and utility.prg .

EXPERT SYSTEM FLOW DIAGRAM

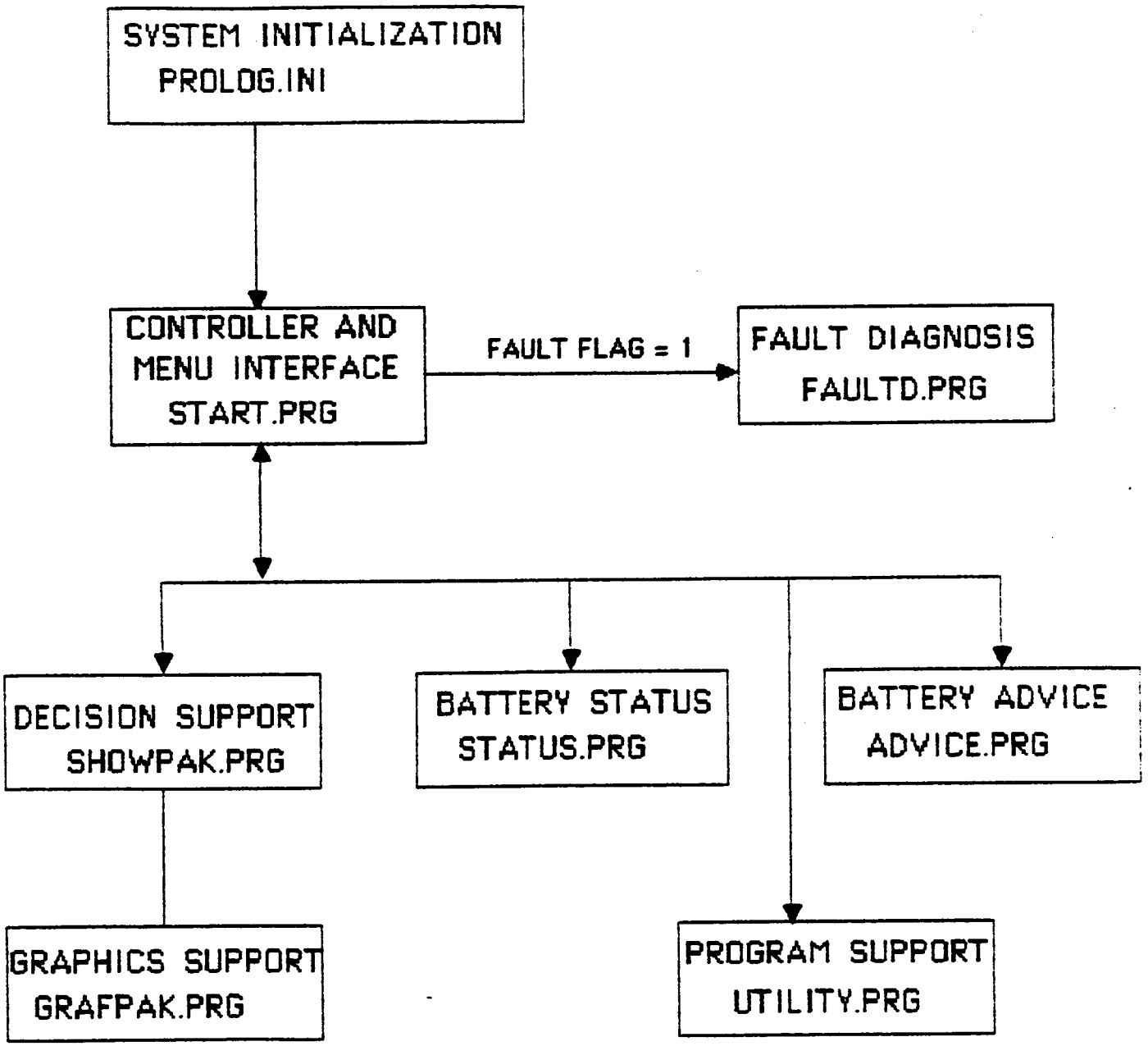


FIGURE 2

2.4.2.2 faultd.prg

Identification: faultd

Function: Perform fault diagnosis for the HST EPS Testbed.

Input: curf2.dat and curf3.dat

Processing: Five conditions are checked to determine the source of the fault. See faultd.doc in Appendix B.

Output: Output is in the form of screen report detailing the fault cause(s) and advising on correctional procedures.

Interfaces: Called by start.prg - fault_diag/0.

2.4.2.3 status.prg

Identification: status

Function: Status analysis is performed for Batteries 1 to 6.

Input: Battery number (Bat), showf3.dat, showf4.dat, showf8.dat, showf13.dat

Processing: Status checks reconditioning flag first. If battery is being reconditioned status stops because data would be misleading. If not, temperature, workload, charging scheme and divergence are checked using averages which are compared to threshold values. See status.doc in Appendix B.

Output: Output is in the form of a screen report detailing the condition of the battery with respect to the above checks.

Interfaces: Called by start.prg - battery_status(Bat).
Calls functions in utility.prg.

2.4.2.4 advice.prg

Identification: advice

Function: Advice uses trend analysis for voltage, recharge ratio, temperature and divergence to give further detail on three subjects: whether a battery needs reconditionin, changes in charging scheme or changes in workload.

Input: Battery number (Bat), Advice Menu Choice (1 to 3), showf1.dat, showf2.dat, showf3.dat, showf5.dat, showf9.dat and showf13.dat.

Processing: Depending on the Choice, data files are read, trends are derived using the difference of two weighting functions and deviation factors. These trends are then compared to conditions to tell whether a battery needs to be changed. Explanations are given to back up the resulting diagnosis. See advice.doc in Appendix B.

Output: Output is in the form of a screen report detailing Battery Advice and explanations.

Interfaces: Called by start.prg - advice(Bat,Choice).
Calls functions in utility.prg.

2.4.2.5 showpak.prg

Identification: showpak

Function: Decision Support portion of the Expert System providing 12 Plots to the user for each battery.

Input: Plot (N), Battery Number (Bat), Orbit number (Orbit), showf#.dat (# from 1 to 12).

Processing: Data from the appropriate data file, for the appropriate Battery is read. The data structure show/11, containing the parameters needed for plotting, is called. See showpak.doc in Appendix B.

Output: Plotting parameters are passed to grafpak.prg

Interfaces: Called by start.prg - show_view(N,Bat,Orbit).
Calls functions in grafpak.prg and utility.prg.

2.4.2.6 grafpak.prg

Identification: grafpak

Function: Draws to the screen the any of the 12 available plots, per battery.

Inputs: List of points to be plotted and all plotting parameters including captions.

Processing: Uses graphics primitives to draw plots on the screen. Plots have X and Y axes, title, header and points displayed in color and symbol. Missing data and data out of range are also displayed. See grafpak.doc in Appendix B for more details and start.doc for a listing of the graphs.

Outputs: Plots drawn to the screen.

Interfaces: Called by showpak.prg - graphplus/6 and plot/8. Calls functions in utility.prg

2.4.2.7 utility.prg

Identification: utility

Function: Collection of miscellaneous Prolog functions used by one or more of the Prolog routines.

Inputs: Parameters are passed for the particular function call.

Processing: Depends on the function call. See utility.doc in Appendix B for detailed description of the functions as well as built-in Arity functions. Details on the handling of data files is also described there.

Outputs: Sends requested values back to the calling function.

Interfaces: Called by start.prg, faultd.prg, status.prg, advice.prg, showpak.prg, grafpak.prg.

2.4.2.8 prolog.ini

Identification: prolog.ini

Function: Consults the programs needed to run the Expert System

Inputs: No inputs.

Processing: start.prg, faultd.prg, status.prg, advice.prg, showpak.prg grafpak.prg and utility.prg are loaded at the initiation of Prolog, when the user enters 'api' at the DOS prompt. All data files are copied to the NICBES directory. The currenet data files and fault.dat are also loaded. The Expert System is then called into operation.

Outputs: No output.

Interfaces: No interfaces.

SECTION 3. ENVIRONMENT

3.1 Equipment Environment.
The following computer equipment is needed for the execution of NICBES:
DEC LSI-11
RS232 cable connector
IBM-PC AT
STAR-SD-15 Printer

3.2 Software Support
The following computer software is needed for the execution of NICBES:
DOS (IBM's operating system)
MICROSOFT C
ARITY PROLOG - Version 4.1

3.3 Data Base
The following paragraphs will detail the data base utilized by NICBES.

3.3.1 General Characteristics.
As NICBES is actually two systems, a data base description will be given for each.
First for the Data-Handler whose data base consists of telemetry, received and processed every one minute. This dynamic data base is not stored, but condensed and summarized by performing mathematical operations. The final historical data will be written to files for use by the Expert System. The only limitations for the telemetry are time constraints and reading and writing validity.

The data base for the Expert System consists of the data output files written by the Data-Handler. These files are static and should not be modified. However, they can be stored in uniquely referrenced locations for later review.

3.3.2 Organization and Detailed Description.
Telemetry for the Data-Handler:
Start of Telemetry Burst
A

Header Information (Integer)
1. year
2. day of year - 198X
3. hour - 0 to 24
4. minute - 0 to 60
5. second - 0 to 60
6. orbit - Positive Integer
7. phase - 0 for discharge, 1 for charge
8. day minute (minute in charge) - 0 to 70
9. night minute (minute in discharge) - 0 to 37

Battery Information (for each of 6 batteries)

10 - 351, 57 values for each battery		
battery number	Integer	1 - 6
cell voltage	23 Reals	-2 to +2 volts
cell pressure	23 Integers	0 to 150 psi
battery voltage	Real	0 to 40 volts
battery current	Real	-30 to +25 amps negative for discharge phase positive for charge phase
bprc current	Real	0 to 5 amps
temperature sensors	6 Reals	-15 to 30 (degrees C)
battery reconditioning	Integer	0 for no, 1 for yes

Miscellaneous Information

352 - 364	Solar Array current	13 Reals	0 to 20 amps
365 - 367	Bus Voltage	3 Reals	0 to 40 volts
368 - 370	Bus current	3 Reals	0 to 90 amps

Reals are five place floating point numbers. Each telemetry value is sent one per line with an associated new line and carriage return. 370 values are sent every one minute, 96 minutes per orbit.

Data Files for the Expert System:
All data files are written in list format. All the show files have 6 lists, one for each battery. All the data files are loaded into the PROLOG Expert System as facts. See documentation in Appendix B for details.

fault.dat - Contains a fault flag = 1 if there was a fault
= 0 if no fault was detected.

curf1.dat - Contains the current orbit number and
a reconditioning flag for each battery = 1 for reconditioning
= 0 no reconditioning.

curf2.dat - Contains Phase (charge or discharge)
Day_min
Current from 13 SPAs (Solar Panel Array)
Current from 3 Busses
Average Temperature for 6 Batteries

curf3.dat - Contains 6 battery cell voltages (23 per battery)

showf1.dat - File contains battery voltage at EOD for last 12 orbits, in chronological order.

showf2.dat - File contains the battery voltage during high in-charge period, last 12 orbits.

showf3.dat - File contains the recharge ratio = AHO/AHI per orbit for 12 orbits.
showf4.dat - File contains cell voltages at EOD, with the high value, low value and average of all values, in this order for each of the last 12 orbits

showf5.dat - File contains cell voltages at high-charge; high, low and average of all values, order H,L,A, for each of last 12 orbits, per battery.

showf6.dat - File contains 23 cell voltages at *EOD* for each battery, from the latest orbit.

showf7.dat - File contains 23 cell voltages at high-charge for each battery, from latest orbit.

showf8.dat - File contains the averages of the six temperature sensors (degrees C), at two minute intervals over the latest orbit. The first value in this file is the minute into orbit, followed by the temperature readings for the batteries.

showf9.dat - File contains the average battery temperatures per orbit for the last 12 orbits.

showf10.dat - File contains the 23 cell pressures taken at EOC and then EOD for each battery in the last full orbit.

showf11.dat - File contains the time on trickle charge for each battery from last 12 orbits.

showf12.dat - File contains battery current during reconditioning, at 2-minute intervals, for last reconditioning of each battery. It is recorded every 2 minutes, only when battery reconditioning is 1 and only for one orbit. The file contains zeroes until a battery is reconditioned.

showf13.dat - File contains AHO summed at EOD over last 12 orbits.

SECTION 4. PROGRAM MAINTENANCE PROCEDURES

4.1 Conventions.

Each routine in the programs comprising the Data-Handler have headers as well as code documentation. The Expert System files are documented in separte files having the same name as the Prolog program but with 'doc' as their extension.

a. Conventional extensions to file names are designed as nmemonic identifiers (file and variable names) based upon descriptive abbreviations of function title.

c	'C' programs
obj	object files
exe	executable files
asm	assembler programs
prg	Prolog programs
dat	data files
doc	document files

b. Refer to SAMSO EX 2.3.3 and MIL-STD-847 (Documentation).

4.2 Verification Procedures.

Any enhancements added to the Data-Handler should be verified by checking the data output files. It is always wise to test the changes on a small test case before running the whole procedure. One enhancement that could be made is to increase the error checking on incoming telemetry so that faults could be detected directly from the Data-Handler. Another enhancement would be to check the ranges on each telemetry value as it is read in.

Changes to the Expert System logic would have to be verified by a Nickel Cadmium Battery 'expert' for validity. These could include changes to the deviation factors, threshold variations and adding conditions to be checked. Enhancements to the screen displays can obviously be checked by running the Expert System and viewing the screen.

The Test Procedures listed in Appendix C will give you a baseline upon which to verify any changes.

4.3 Error Conditions.

There are no special provisons for operating system errors. Procedures to take at such instances would include rebooting the IBM-PC AT, checking to see that all files are intact and starting the NICBES system again.

4.4 Special Maintenance Procedures.

Data files written by the Data-Handler need to be archived for later use. One way of doing this would be to create a data directory at the root. Then for each set of data files created, a sub-directory could be created into which the data file set could be copied. This sub-directory can then be referrenced by location and time. A command file - data.bat, has been written for just this purpose. It is located in C:\USR. To run data.bat simply enter 'data directory-name <CR>'. The directory-name can be a date as 01-16-87 for later referencing.

It is also wise to make periodic backups of the NICBES system as well as the accumulated data. There is no need to backup the MICROSOFT C directories (except C:\USR) or the ARITY PROLOG files as these can always be re-installed from their original disks.

4.5 Special Maintenance Programs.
There are no special maintenance programs.

4.6 Listings.
All NICBES program listings will accompany this Maintenance Manual. The 'C' programs are documented internally while the Prolog programs are documented in files with the same name as the Prolog routine but with 'doc' as their extension. APPENDIX A contains the Data-Handler listings. APPENDIX B contains the Expert System listings.

APPENDIX A

DOCUMENTED CODE LISTINGS FOR THE DATA-HANDLER

HST. H

```
/*    Define Data-Handler constants                                    */
#define         SUCCESS          1
#define         FAIL             0

#define         EOC              2          /* Phase signals        */
#define         EOD              3
#define         EVMIN            1
#define         DEFLT        -9999.0        /* Missing data signal*/

#define           SHOWF1DAT  1              /* Numbers for datafiles*/
#define           SHOWF2DAT  2
#define           SHOWF3DAT  3
#define           SHOWF4DAT  4
#define           SHOWF5DAT  5
#define           SHOWF6DAT  6
#define           SHOWF7DAT  7
#define           SHOWF8DAT  8
#define           SHOWF9DAT  9
#define           SHOWF10DAT 10
#define           SHOWF11DAT 11
#define           SHOWF12DAT 12
#define           SHOWF13DAT 13
#define           CURF1DAT   14
#define           CURF2DAT   15
#define           CURF3DAT   16
#define           FAULTDAT   17

#include      <stdio.h>                  /* Include files for D-H */
#include      <errno.h>
#include      <stdlib.h>
#include      <process.h>
#include      <time.h>
#include      <signal.h>

/* *****       Structures for HST incomming data file      *****    */

struct      bati      {
      int         battno;          /* Battery Number          */
      float       cellv[23];       /* Cell Voltage            */
      float       cellp[23];       /* Cell Pressure           */
      float       batv;            /* Battery Voltage         */
      float       batc;            /* battery Current         */
      float       bprcc;           /* BPRC Current            */
      float       batemp[6];       /* Battery Temperature     */
      int         batrecond;       /* Battery Reconditioning*/
};
```

```
struct     bus                  /* Bus data                    */
{
     float      busv;           /* Bus Voltage                 */
     float      busc;           /* Bus Current                 */
};

struct     hstid                    /* HST Incomming Data      */
{
     int       year;             /* Year                        */
     int       day;              /* Day                         */
     int       hour;             /* Hour                        */
     int       min;              /* Minutes                     */
     int       sec;              /* Seconds                     */
     int       orbit;            /* Orbit#                      */
     int       phase;            /* Phase=0 if night            */
                                 /*       =1 if day             */
     int       day_min;          /* Min into charge peroid*/
     int       night_min;        /* Min into discharge  " */
     struct  bati  batd[6];      /* Battery Data                */
     float   spac[13];           /* SPA Current                 */
     struct  bus   bd[3];        /* Bus info for three bus*/
}hid[2];


/*     Show File#1(12 orbits for a battery and for 6 batteries)  */
float     eod_voltage[6][12];    /* Voltage taken at EOD for 12  */
                                 /* orbits, 6 batteries.  To be  */
                                 /* updated when phase changes   */
                                 /* from 0 to 1                  */

/*      Show File#2(12 orbits for a battery and for 6 batteries) */
float     hc_voltage[6][12];     /* Max Batt Voltage taken for 12*/
float     high_buffer[6];        /* orbits, 6 batteries during   */
                                 /* phase 1.  To be updated when */
                                 /* phase changes from 1 to 0    */

/*      Show File#3(12 orbits for a battery and for 6 batteries) */
float     rc_ratio[6][12];       /* Recharge Ratio for 12 orbits,*/
float     ahoo[6];               /* 6 batteries.  Sum of battery */
float     ahi[6];                /* current during phase 1       */
                                 /* divided by sum of battery    */
                                 /* current during phase 0.  It  */
                                 /* is always > 1.  Updated when */
                                 /* phase changes from 0 to 1    */
```

```
/*      Show File#4                                                    */

float    cv_eod_hv[6][12];      /* Cell Voltage for 6 batteries       */
float    cv_eod_lv[6][12];      /* Calculate high, low and avg        */
float    cv_eod_av[6][12];      /* out of 23 cells when phase         */
                                /* changes to 1.                      */

/*      Show File#5                                                    */
float    cv_hc_hv[6][12];       /* Calculate high, low and avg        */
float    cv_hc_lv[6][12];       /* out of 23 cells during phase       */
float    cv_hc_av[6][12];       /* 1, at high-charge. To be           */
                                /* update when phase changes          */
                                /* from 1 to 0.                       */

/*      Show File#6                                                    */
float     cellv_eod[6][23];     /* Cell Voltage taken at EOD.         */
                                /* To be updated when phase           */
                                /* changes from 0 to 1.               */

/*      Show File#7                                                    */
float     cellv_hc[6][23];      /* Cell Voltage taken during          */
                                /* phase 1 at high-charge.            */
                                /* To be updated when phase           */
                                /* change from 1 to 0                 */

/*      Show File#8                                                    */
float     avgt[6][48];          /* avg temperature taken from         */
                                /* 6 sensors per battery, 6           */
                                /* batteries, at 2 min intervals      */

/*      Show File#9                                                    */
float     avg_temp[6][12];      /* Every Minute take an average       */
float     avg_temp_buffer[6];   /* of 6 sensors' temp for 96          */
                                /* minutes in the orbit, for 12       */
                                /* orbits,& for each of 6 batts       */
                                /* Avg over 96 minutes or entire      */
                                /* orbit.  To be updated when         */
                                /* when phase change from 1 to 0      */

/*      Show File#10                                                   */
float     cp_eod[6][23];        /* Cell pressure at eod               */
float     cp_eoc[6][23];        /* Cell pressure at eoc               */
```

```
/*      Show File#11                                                */
float     time_tc[6][12];       /* When battery current is lower*/
int       trickle[6];           /* than 3 amps, add 1 to trickle*/
                                /* time counter.  Only during   */
                                /* charge phase.                */

/*      Show File#12                                                */
int       rc_orbit[6];          /* Orbit at which reconditioning*/
float     bc_drc[6][48];        /* takes place.  Battery current*/
                                /* during reconditioning at     */
                                /* 2 minute intervals           */

/*      Stat file#1                                                 */
float     aho[6][12];           /* Keep running sum of amp hours*/
                                /* out during phase 0.          */

/*      Cur File #2                                                 */
float     batt_avgt[6];         /* Average temp of six sensors  */
                                /* for each of 6 batteries.     */

/*      fault.data                                                  */
int     fault[2];               /* Fault flag, fault[0]=1 fault */
                                /*                    =0 no fault*/
                                /* fault[1] = -1 when problem   */
                                /* with communication           */
```

DATA-HDL.C

```
#define CILIMIT  5

#include "hst.h"

int conseq_incmplt = 0;
/*****************************************************************/
main(argc,argv)     /* main is the Data-Handler driver.  It controls the*/
int argc;           /* program flow and determines the times for events */
char **argv;
{
     int orbitno = 0,err;
     int phase1,phase2;                              /* phase1 = last phase   */
                                                     /* phase2 = current phase*/
     int phase_signal;                               /* EOD phase from 0 to 1 */
                                                     /* EOC phase from 1 to 0 */
     init();                                         /* initialization call   */
     while(1)
     {
          conseq_incmplt = 0;
          while ((err = read_data()) == FAIL) /* Read telemetry burst  */
               incomplete();                  /* Check for consecutive */
                                              /* incmplt data bursts   */
          if ((err = check_fault()) == FAIL)  /* Check for faults      */
          {                                   /* If fault set fault    */
               fault[0] = 1;                  /* flag = 1 and finish   */
               wf(FAULTDAT);
               finish();
          }

                                              /* determine phase change*/
          if((phase1 = hid[0].phase) == (phase2 = hid[1].phase))
                    phase_signal = EVMIN;
          if (!phase1 && phase2) phase_signal = EOD;
          if (phase1 && !phase2) phase_signal = EOC;

                                              /* EOD and EOC cases     */
          if ((phase_signal == EOD) || (phase_signal == EOC))
          {
               process(phase_signal, orbitno);/* process data EOC/EOD  */
               if(phase_signal == EOC)
                    printf("%d.  ORBIT = %d\n",++orbitno,hid[0].orbit);
          }
                                              /* EVMIN case            */
          process(EVMIN,orbitno);             /* process data EVMIN    */
     } /* end while loop */
     finish();                                /* exiting routine       */
} /*  end main */
```

```
/*****************************************************************************/
init()              /* Sets the communication port and signal, initializes */
{                   /* interrupt handler, global buffers and telemetry read*/
     int err,port;
     int sig_catch();

     signal(SIGINT,sig_catch);                    /* initialize user input */
                                                  /* signal catcher        */
     serini();                                    /* initialize interrupt  */
                                                  /* handler               */
     port = set_port((int) 0);                    /* set comm port for RS232*/
     df_init();                                   /* initialize data buffers*/
     read_init();                                 /* initialize telemetry  */
                                                  /* reading to orbit start*/
} /* end init */

/*****************************************************************************/
read_init()         /* Initializes telemetry reading to first full orbit.*/
{
     int i,err;

     i = DEFLT;
     while(i != 1)                                /* Orbit starts at night */
     {                                            /* minute 1, discharge   */
          while ((err = read_data()) == FAIL)     /* phase.  No action til */
               incomplete();                      /* start found.Checks for*/
          i = hid[1].night_min;                   /* consecutive incomplete*/
     }                                            /* telemetry bursts.     */
     printf("Starting first full orbit\n");
     process(EVMIN,0);                            /* Process first min data*/
} /* end read_init */

/*****************************************************************************/
finish()                        /* Exits Data Handler and writes files.     */
{
     serrst();                                    /* end interrrupt handler*/
     write_file(EOC);                             /* write output files    */
     write_file(EOD);
     exit(0);                                     /* exit Data-Handler     */
} /* end finish */
```

ORIGINAL PAGE IS
OF POOR QUALITY

```
/*****************************************************************************/
sig_catch()                         /* Catches '^C' signal input by operator to */
                                    /* halt Data-Handler.                        */
{
    printf("Interrupt caught; exiting!\n");
    finish();
} /* end sig_catch */
/*****************************************************************************/
process(phase_signal,orbitno)       /* Calls the routines necessary to */
int phase_signal, orbitno;          /* process the telemetry.          */
{
    process_data(phase_signal,orbitno);     /* process data            */
    write_file(phase_signal);               /* write files             */
    df_init1(phase_signal);                 /* reintialize buffers     */
} /* end process */
/*****************************************************************************/
incomplete()          /* counts no. of consecutive incomplete data bursts */
{
/*  printf("consecutive = %d\n",conseq_incmplt); */
    conseq_incmplt++;                       /* inc consecutive  in-     */
    if (conseq_incmplt == CILIMIT)          /* complete telemety runs*/
    {                                       /* if CILIMIT runs, halt */
        printf("Recieved %d consecutive incomplete\n",CILIMIT);
        printf("telemetry bursts.  Shutting down!!\n");
        fault[0] = 1;                       /* Set fault flags       */
        fault[1] = -1;
        wf(FAULTDAT);
        finish();
    }
} /* end incomplete */
```

```
#define T1LIMIT  2
#define T2LIMIT  180

#include "hst.h"

char buf[2048];
int count;
float cell[23];
/*************************************************************************/
read_data()   /* Read telemetry from DEC LSI-11 over RS232 every 1 min.*/
{
     int err;
/*  Call proc_sync to read start character 'A',  call proc_head to read*/
/*  header info, proc_bat() to read battery data, and proc_solar to    */
/*  to read SPA and bus data                                           */
     proc_sync();
     count = 0;
     if ((err = proc_head()) == FAIL) return(FAIL);
     if ((err = proc_bat()) == FAIL) return(FAIL);
     if ((err = proc_solar()) == FAIL) return(FAIL);
     return(SUCCESS);
} /* end read_data */

/*************************************************************************/
proc_sync()    /* Synchronize reading data to start of telemetry burst.*/
{
     char cc;

     while (cc = get2())                          /* Get character from    */
     {                                            /* interrupt hdlr buffer */
          if (cc == 'A')                          /* If 'A',telemetry start*/
          {
/*             printf("Sync Received!\n"); */
               break;
          }
          if(cc == 'B')                           /* If 'B', shutdown      */
          {
               printf("\n\nReceived shutdown signal from DEC LSI-11!\n");
/*             fault[0] = 1;
               fault[1] = ?;
               wf(FAULTDAT);  */
               finish();
          }
     } /* end while loop */

     cc = get1();                                 /* Read newline and CR   */
     cc = get1();                                 /* to position pointer at*/
} /* end proc_sync */                            /* next character        */
```

```
/*************************************************************************/
proc_head()                         /* Read header data from input buffer.  */
{
     int err;

     if ((err = read_buffer(9)) == FAIL)        /* Read 9 header items    */
          return(FAIL);

     sscanf(buf,"%d%d%d%d%d%d%d%d%d",           /* Put header data in buf*/
          &hid[1].year,&hid[1].day,             /* into structured array */
          &hid[1].hour,&hid[1].min,             /* hid[1] in int format  */
          &hid[1].sec,&hid[1].orbit,&hid[1].phase,
          &hid[1].day_min,&hid[1].night_min);
                                                 /* Print header data     */
/*   printf("time=%d\nday=%d\nhour=%d\nmin=%d\nsec=%d\norbit=%d\nphase=%d\n,
          day_min=%d\nnight_min=%d\n\n",
          hid[1].year,hid[1].day,hid[1].hour,hid[1].min,hid[1].sec,
          hid[1].orbit,hid[1].phase,hid[1].day_min,hid[1].night_min); */

     return(SUCCESS);
} /* end proc_head */
```

```
/*************************************************************************/
proc_bat()                      /* Read battery data from input buffer.  */
{
     int j,k,err;

     for(k=0;k<6;k++)                          /* For 6 batteries       */
     {
          if ((err = read_buffer(1)) == FAIL) /* Read battery no       */
               return(FAIL);

          sscanf(buf,"%d",&hid[1].batd[k].battno);     /* Put buf       */
                                                /* contents into hid[1]  */
/*        printf("batt no = %d, count = %d\n",hid[1].batd[k].battno,count); *

          if ((err = read_buffer(23)) == FAIL)/* Get 23 cell voltages  */
               return(FAIL);

          sscanf(buf,"%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f",
               &cell[0],&cell[1],&cell[2],&cell[3],&cell[4],&cell[5],
               &cell[6],&cell[7],&cell[8],&cell[9],&cell[10],&cell[11],
               &cell[12],&cell[13],&cell[14],&cell[15],&cell[16],&cell[17],
               &cell[18],&cell[19],&cell[20],&cell[21],&cell[22]);

          for (j=0;j<23;j++)
               hid[1].batd[k].cellv[j] = cell[j];

          if ((err = read_buffer(23)) == FAIL) /* Get 23 cell pressures */
               return(FAIL);

          sscanf(buf,"%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f",
               &cell[0],&cell[1],&cell[2],&cell[3],&cell[4],&cell[5],
               &cell[6],&cell[7],&cell[8],&cell[9],&cell[10],&cell[11],
               &cell[12],&cell[13],&cell[14],&cell[15],&cell[16],&cell[17],
               &cell[18],&cell[19],&cell[20],&cell[21],&cell[22]);

          for (j=0;j<23;j++)
               hid[1].batd[k].cellp[j] = cell[j];

          if ((err = read_buffer(10)) == FAIL)/* Get remaining 10 bat  */
               return(FAIL);

          sscanf(buf,"%f%f%f%f%f%f%f%f%f%d",  /* Put buf contents      */
               &hid[1].batd[k].batv,          /* into hid[1]           */
               &hid[1].batd[k].batc,&hid[1].batd[k].bprcc,
               &hid[1].batd[k].batemp[0],&hid[1].batd[k].batemp[1],
               &hid[1].batd[k].batemp[2],&hid[1].batd[k].batemp[3],
               &hid[1].batd[k].batemp[4],&hid[1].batd[k].batemp[5],
               &hid[1].batd[k].batrecond);
     }     /* end for loop, battery 1 to 6  */
     return(SUCCESS);
} /* end proc_bat */
```

```
/*************************************************************************/
proc_solar()                /* Read SPA and bus data from input buffer.  */
{
     int j,err;

     if ((err = read_buffer(13)) == FAIL)      /* Get 13 SPA values      */
               return(FAIL);                   /* values from buffer     */

                                               /* Put buf contents into */
     sscanf(buf,"%f%f%f%f%f%f%f%f%f%f%f%f%f", /* hid[1]                 */
          &hid[1].spac[0],&hid[1].spac[1],&hid[1].spac[2],
          &hid[1].spac[3],&hid[1].spac[4],&hid[1].spac[5],
          &hid[1].spac[6],&hid[1].spac[7],&hid[1].spac[8],
          &hid[1].spac[9],&hid[1].spac[10],&hid[1].spac[11],
          &hid[1].spac[12]);

     if ((err = read_buffer(6)) == FAIL)       /* 3 busses, volt and     */
          return(FAIL);                        /* current for each       */

     sscanf(buf,"%f%f%f%f%f%f",                /*  Put buf contents      */
          &hid[1].bd[0].busv,                  /*  into hid[1]           */
          &hid[1].bd[0].busc,&hid[1].bd[1].busv,&hid[1].bd[1].busc,
          &hid[1].bd[2].busv,&hid[1].bd[2].busc);

/*   printf("solar count = %d\n",count); */
/*   for(j=0;j<3;j++) */                       /* Print bus data         */
/*   {
          printf("bd[%d].busv = %f\n",j,hid[1].bd[j].busv);
          printf("bd[%d].busc = %f\n",j,hid[1].bd[j].busc);
     } */

     return(SUCCESS);
} /* end proc_solar */
```

```
/*************************************************************************/
read_buffer(k)              /* Retrieves telemetry values from interrupt*/
int  k;                     /* handler buffer and puts them in buf[]    */
{
     int i=0, nl_count = 0;

     while(nl_count != k)                          /* For k data points     */
     {
          if ((buf[i] = getl()) == EOF)            /* Get each character    */
               return(FAIL);                       /* from input buffer and */

          if(buf[i] == '\r')                       /* put in buf, including */
          {                                        /* newline and CR        */
               count++;                            /* count values read per */
                                                   /* telemetry run         */
               nl_count++;                         /* Returns FAIL if EOF   */
               if ((buf[i] = getl()) == EOF)       /* read before end of    */
                    return(FAIL);                  /* telemetry burst       */
          }
          i++;
     }
     buf[i] = '\0';                                /* sets end of data in   */
     return(SUCCESS);                              /* in buf[i]             */
} /* end read_buffer */

/*************************************************************************/
getl()                  /* get character from interrupt handler buffer   */
{
     char cc;
     long t0,tn,timer;

     time(&t0);                                    /* initial time          */
     while ((cc = b_char()) == EOF)                /* get char from interrupt*/
     {                                             /* handler buffer, EOFs  */
                                                   /* start timer = tn - t0 */
          timer = time(&tn) - t0;                  /* tn is current time    */
/*        printf("get:  count = %d, timer = %ld\n",count,timer); */
          if ((count < 370) && (timer > T1LIMIT))  /* check for incmplt*/
               return(EOF);                        /* run, T1LIMIT exceeded */
     }
     return(cc);
} /* end get */
```

```
/*************************************************************************/
get2()                  /* get character from interrupt handler buffer   */
{
     char cc;
     long t0,tn,timer;

     time(&t0);                                        /* initial time          */
     while ((cc = b_char()) == EOF)                    /* read char from interrupt*/
     {                                                 /* handler buffer, EOFs  */
                                                       /* start timer = tn - t0 */
          timer = time(&tn) - t0;                      /* tn is current time    */
          if (timer > T2LIMIT)                         /* Exit if T2LIMIT is    */
          {                                            /* exceeded => No Commun.*/
               printf("No communication for 3 minutes;  exiting!\n");
               fault[1] = -1;                          /* write fault flags     */
               fault[0] = 1;
               wf(FAULTDAT);
               finish();
          }
     }
     return(cc);
} /* end get */
```

```
#define  DCHGLIMIT  30
#define  CHGLIMIT   50

#include "hst.h"

int no_druns,no_cruns;
/*************************************************************************/
df_init()    /* Initialization of buffers*/
{
     int     i,j;

/*   printf("in df_init\n"); */
     fault[0] = 0;                              /* FAULTDAT buffer       */
     fault[1] = 0;                              /* FAULTDAT buffer       */
     no_cruns = no_druns = 0;                   /* run counters          */

     for (i=0;i<6;i++)                          /* for 6 batteries       */
     {
          batt_avgt[i] = DEFLT;                 /* CURF2DAT buffer       */
          avg_temp_buffer[i] = 0.0;             /* SHOWF9DAT work buffer */
          high_buffer[i] = DEFLT;               /* SHOWF2DAT work buffer */
          ahoo[i] = 0.0;                        /* SHOWF3DAT work buffer */
          ahi[i] = 0.0;                         /* SHOWF3DAT work buffer */
          rc_orbit[i] = 0;                      /* SHOWF12DAT buffer     */
          trickle[i] = 0;                       /* SHOWF11DAT work buffer*/

          for (j=0;j<12;j++)                    /* for 12 orbits         */
          {
               eod_voltage[i][j] = DEFLT;       /* SHOWF1DAT buffer      */
               hc_voltage[i][j] = DEFLT;        /* SHOWF2DAT buffer      */
               rc_ratio[i][j] = DEFLT;          /* SHOWF3DAT buffer      */
               cv_eod_hv[i][j] = DEFLT;         /* SHOWF4DAT buffer      */
               cv_eod_hv[i][j] = DEFLT;         /* SHOWF4DAT buffer      */
               cv_eod_av[i][j] = DEFLT;         /* SHOWF4DAT buffer      */
               cv_hc_hv[i][j] = DEFLT;          /* SHOWF5DAT buffer      */
               cv_hc_lv[i][j] = DEFLT;          /* SHOWF5DAT buffer      */
               cv_hc_av[i][j] = DEFLT;          /* SHOWF5DAT buffer      */
               avg_temp[i][j] = DEFLT;          /* SHOWF9DAT buffer      */
               aho[i][j] = DEFLT;               /* SHOWF13DAT buffer      */
               time_tc[i][j] = DEFLT;           /* SHOWF11DAT buffer     */
          }
          for (j=0;j<23;j++)                    /* for 23 cells per batt */
          {
               cellv_eod[i][j] = DEFLT;         /* SHOWF6DAT buffer      */
               cellv_hc[i][j] = DEFLT;          /* SHOWF7DAT buffer      */
               cp_eod[i][j] = DEFLT;            /* SHOWF10DAT buffer     */
               cp_eoc[i][j] = DEFLT;            /* SHOWF10DAT buffer     */
          }
```

```
            for (j=0;j<48;j++)
            {
                avgt[i][j] = DEFLT;              /* SHOWF8DAT buffer      */
                bc_drc[i][j] = 0.0;              /* SHOWF12DAT buffer     */
            }
        }
        wf(FAULTDAT);                            /* write no fault yet    */
} /*end df_init */
```

```
/*************************************************************************/
df_initl(phase_signal)  /* After each telemetry burst is read, or after*/
                        /* EOC or EOD, buffers used for processing need*/
int       phase_signal;  /* to be re-initialized.                       */
{
     int      i,j;

/*   printf("in df_initl \n"); */
switch(phase_signal)                              /* start switch        */
{
case EOC:                                         /* EOC case            */
     no_cruns = no_druns = 0;                     /* run counters        */
     for (i=0;i<6;i++)
     {
          avg_temp_buffer[i] = 0.0;               /* SHOWF9DAT work buffer */
          high_buffer[i] = DEFLT;                 /* SHOWF2DAT work buffer */
          ahi[i] = 0.0;                           /* SHOWF3DAT work buffer */
          ahoo[i] = 0.0;                          /* SHOWF3DAT work buffer */

          for (j=0;j<23;j++)                      /* for 23 cells per batt */
          {
               cellv_hc[i][j] = DEFLT;            /* SHOWF7DAT buffer      */
               cp_eoc[i][j] = DEFLT;              /* SHOWF10DAT buffer     */
               cp_eod[i][j] = DEFLT;              /* SHOWF10DAT buffer     */
               cellv_eod[i][j] = DEFLT;           /* SHOWF6DAT buffer      */
          }
          for (j=0;j<48;j++)
              avgt[i][j] = DEFLT;                 /* SHOWF8DAT buffer      */
     }
     break;                                       /* end EOC case        */
                                                  /* No EOD case         */

case EVMIN:                                       /* Every Minute case   */
     for (i=0;i<6;i++)                            /* for 6 batteries     */
          batt_avgt[i] = DEFLT;                   /* CURF2DAT buffer     */
     hidtohid();                                  /* switch hid columns  */
     break;                                       /* end EVMIN case      */
} /* end switch */
} /* end df_initl */
```

```
/*************************************************************************/
hidtohid()  /* After each telemetry burst is read and processed, the   */
            /* data in hid[1] is put in hid[0] to prepare for next     */
            /* data burst which will be read into hid[1].              */
{
     int i,j;

/*   printf("in hidtodhid, ready to exchange buffers\n"); */
     hid[0].year = hid[1].year;                  /* switch header items   */
     hid[0].day = hid[1].day;
     hid[0].hour = hid[1].hour;
     hid[0].min = hid[1].min;
     hid[0].sec = hid[1].sec;
     hid[0].orbit = hid[1].orbit;
     hid[0].phase = hid[1].phase;
     hid[1].phase = DEFLT;
     hid[0].day_min = hid[1].day_min;
     hid[1].day_min = DEFLT;
     hid[0].night_min = hid[1].night_min;

     for (i=0;i<6;i++)                           /* switch battery items  */
     {
          hid[0].batd[i].battno = hid[1].batd[i].battno;
          for (j=0;j<23;j++)                     /* for 23 cells per batt */
          {
               hid[0].batd[i].cellv[j] = hid[1].batd[i].cellv[j];
               hid[1].batd[i].cellv[j] = DEFLT;
               hid[0].batd[i].cellp[j] = hid[1].batd[i].cellp[j];
          }
          hid[0].batd[i].batv = hid[1].batd[i].batv;
          hid[0].batd[i].batc = hid[1].batd[i].batc;
          hid[0].batd[i].bprcc = hid[1].batd[i].bprcc;
     }
     for (i=0;i<13;i++)                          /* switch SPA items      */
          hid[0].spac[i] = hid[1].spac[i];
          hid[1].spac[i] = DEFLT;

     for (i=0;i<3;i++)                           /* switch bus items      */
     {
          hid[0].bd[i].busv = hid[1].bd[i].busv;
          hid[0].bd[i].busc = hid[1].bd[i].busc;
          hid[1].bd[i].busc = DEFLT;
     }
} /* end hidtohid */
```

```
/*************************************************************************/
move_buffers(phase_signal)  /* Prepares the global   buffers for move  */
int     phase_signal;
{

/*   printf("in move_buffers, ready to move buffers\n"); */
switch(phase_signal)                          /* start switch          */
{
case EOC:                                     /* start EOC case        */
          move(hc_voltage);                   /* SHOWF2DAT buffer      */
          move(avg_temp);                     /* SHOWF9DAT buffer      */
          move(time_tc);                      /* SHOWF11DAT buffer     */
          move(cv_hc_hv);                     /* SHOWF5DAT buffer      */
          move(cv_hc_lv);
          move(cv_hc_av);
          break;                                /* end EOC case          */

case EOD:                                     /* start EOD case        */
          move(eod_voltage);                  /* SHOWF1DAT buffer      */
          move(rc_ratio);                     /* SHOWF3DAT buffer      */
          move(aho);                          /* SHOWF13DAT buffer      */
          move(cv_eod_hv);                    /* SHOWF4DAT buffer      */
          move(cv_eod_lv);
          move(cv_eod_av);
          break;                              /* end EOD case          */
                                              /* no EVMIN case         */
} /* end switch */
} /* end move_buffers() */

/*************************************************************************/
move(buffer)  /* If the Data-Handler contines after 12 completed orbits*/
              /* the buffers(6,12) will lose their first column, the   */
              /* remaining data will be shifted one column to the left,*/
              /* the next orbits data will be put in the 12th column.  */

float     buffer[6][12];
{
     int       i,j,jj;

     for(i=0;i<6;i++)                         /* for 6 battery rows    */
     {
          for (j=0;j<10;j++)                  /* for 11 orbit columns  */
          {                                   /* column 1 is dropped   */
               jj = j+1;                      /* shift each column one */
               buffer[i][j] = buffer[i][jj];  /* column to the left    */
          }
          buffer[i][11] = DEFLT;              /* initialize 12th column*/
     }
} /* end move */
```

```
/*************************************************************************/
process_data(phase_signal,orbitno)  /* Perform mathematical operations */
                                    /* to summarize telemetry. Keeps   */
int     phase_signal, orbitno;      /* results in global buffers.      */
{
     int    current_min,col;
     int    jj,i,j;
     float  x,x1,x2,x3;
     float  sum,avg;

/*   printf("in process_data\n"); */
     if (orbitno > 11)
     {
          col = 11;
          move_buffers(phase_signal);
     }else col = orbitno;

switch(phase_signal)                              /* start switch          */
{
case EOC:                                         /* start EOC case        */
     jj = no_cruns + no_druns;
     if (jj < (CHGLIMIT + DCHGLIMIT))             /* Enough runs?          */
        break;

     for (i=0;i<6;i++)                            /* for 6 batteries       */
     {
          avg_temp[i][col] = avg_temp_buffer[i]/jj; /* SHOWF9DAT buffer*/

          if (ahoo[i] != 0.0)                     /* SHOWF3DAT buffer      */
               rc_ratio[i][col] = ahi[i]/ahoo[i];

          if (trickle[i] != 0)                    /* SHOWF11DAT buffer     */
               time_tc[i][col] = trickle[i];

          for(j=0;j<23;j++)                       /* for 23 cells per batt */
               cp_eoc[i][j] = hid[0].batd[i].cellp[j];
                                                  /* SHOWF10DAT buffer     */
     }
     break;                                       /* end EOC case          */
```

```
case EOD:                                          /* start EOD case        */
     if (no_druns < DCHGLIMIT)                     /* Enough discharge runs?*/
          break;

     for(i=0;i<6;i++)                              /* for 6 batteries       */
     {
          eod_voltage[i][col] = hid[0].batd[i].batv;
                                                   /* SHOWF1DAT buffer      */
          aho[i][col] = ahoo[i];                   /* SHOWF13DAT buffer     */

          x1 = DEFLT;                              /* high, low, avg of cell*/
          x2 = -DEFLT;                             /* voltage at EOD        */
          x3 = 0.0;
          for(j=0;j<23;j++)                        /* for 23 cells per batt */
          {
               x = hid[0].batd[i].cellv[j];
               if (x > x1) x1 = x;
               if (x < x2) x2 = x;
               x3 = x3 + x;
               cellv_eod[i][j] = hid[0].batd[i].cellv[j];
                                                   /* SHOWF6DAT buffer      */
               cp_eod[i][j] = hid[0].batd[i].cellp[j];
          }                                        /* SHOWF10DAT buffer     */
          cv_eod_hv[i][col] = x1;                  /* SHOWF4DAT buffer      */
          cv_eod_lv[i][col] = x2;
          cv_eod_av[i][col] = x3 / 23.0;
     }
     break;                                        /* end EOD case          */
```

```
case EVMIN:                                     /* start EVMIN case      */
     current_min = hid[1].night_min + hid[1].day_min;
     jj = -1;
     if (!(current_min %  2)) jj = current_min/2;

     for (i=0;i<6;i++)                          /* for 6 batteries       */
     {
          sum = 0.0;
          for (j=0;j<6;j++)                     /* for 6 batt temp sensors*/
               sum = sum + hid[1].batd[i].batemp[j];

          avg = sum / 6.0;                      /* avg temp per battery  */
          batt_avgt[i] = avg;                   /* CURF2DAT buffer       */
          avg_temp_buffer[i] = avg_temp_buffer[i] + avg;
                                                /* SHOWF9DAT buffer      */
          if (jj != -1)                         /* every 2 minutes       */
          {
               avgt[i][jj] = avg;               /* SHOWF8DAT buffer      */
               if (hid[1].batd[i].batrecond)    /* SHOWF12DAT buffer     */
               {
                    if ((rc_orbit[i] != hid[1].orbit) &&
                                (rc_orbit[i] != 0))
                    {
                         rc_orbit[i] = hid[1].orbit;
                         for (j=0;j<48;j++)
                              bc_drc[i][j] = 0.0;
                    }
                    bc_drc[i][jj] = hid[1].batd[i].batc;
               }
          }
     }
```

```
    if (hid[1].phase)                          /* CHARGE PHASE          */
    {
         no_cruns++;                           /* increment charge runs */
         for (i=0;i<6;i++)                     /* for 6 batteries       */
         {
              x1 = high_buffer[i];             /* high, low, avg of cell*/
              x2 = -DEFLT;                     /* volts at high-charge  */
              x3 = 0.0;
              for(j=0;j<23;j++)                /* for 23 cells per batt*/
              {
                   x = hid[1].batd[i].cellv[j];
                   if (x > x1) x1 = x;
                   if (x < x2) x2 = x;
                   x3 = x3 + x;
              }
              cv_hc_lv[i][col] = x2;           /* SHOWF5DAT buffer      */
              if (high_buffer[i] < x1)
              {
                   cv_hc_hv[i][col] = x1;
                   cv_hc_av[i][col] = x3 / 23.0;
                   hc_voltage[i][col] = hid[1].batd[i].batv;
                                               /* SHOWF2DAT buffer      */

                   for (j=0;j<23;j++)          /* SHOWF7DAT buffer      */
                        cellv_hc[i][j] = hid[1].batd[i].cellv[j];
              }
              x = hid[1].batd[i].batc;
              ahi[i] = ahi[i] + (x/60.0);      /* SHOWF3DAT work buffer */
              printf("ahi[%d] = %f\n",i,ahi[i]);
              if (x < 3.0)
                   trickle[i] = trickle[i]++;/* SHOWF11DAT work buffer*/
         }                                     /* end CHARGE case       */
    }else{
         if (!hid[1].phase)                    /* DISCHARGE PHASE       */
         {
              no_druns++;                      /* incr discharge runs   */
              for(i=0;i<6;i++)                 /* for 6 batteries       */
                   ahoo[i] = ahoo[i] - (hid[1].batd[i].batc/60.0);
         }                                     /* SHOWF3DAT work buffer */
    }                                          /* end DISCHARGE case    */
    break;                                     /* end EVMIN case        */

} /* end switch */
} /* end process_data() */
```

```
/*************************************************************************/
check_fault()                          /* check telemetry for a fault */
{
     int i,j;
     float x,sum;

     for (i=0;i<13;i++)
     {
          x = hid[1].spac[i];
          if ((hid[1].day_min < 5) && (x < 5.0))
               return(FAIL);
          j = (i + 1)/2;
          if (j)
          {
               if (x >= 16.0) return(FAIL);
          }else{
               if (x >= 8.0) return(FAIL);
          }
          if (!hid[1].phase && (x > 5.0)) return(FAIL);
     }

     for (i=0;i<6;i++)
     {
          sum = 0.0;
          for (j=0;j<6;j++)
          {
               sum = sum + hid[1].batd[i].batemp[j];
          }
          if ((sum > 25.0) || (sum < -10)) return(FAIL);

          for (j=0;j<23;j++)
          {
               x = hid[1].batd[i].cellv[j];
               if ((x <= 0.0) || (x > 1.55)) return(FAIL);
           }
     }

     sum = 0.0;
     for (i=0;i<3;i++);
     {
          x = hid[1].bd[i].busc;
          sum = sum + x;
          if(!hid[1].phase && (x < 5.0)) return(FAIL);
     }
     if (sum > 99.0) return(FAIL);
     return(SUCCESS);
} /* end check_fault */
```

```
#include "hst.h"

/*****************************************************************/
write_file(phase_signal)          /* Determines which data files should */
int     phase_signal;             /* be written at this time.           */
{
     int err;

/*   printf("at write_file\n"); */
switch(phase_signal)                              /* start switch       */
{
case EOC:                                         /* start EOC case     */
     if (!(err = wf(SHOWF2DAT))) printf("Couldn't open %s\n","showf2.dat");
     if (!(err = wf(SHOWF3DAT))) printf("Couldn't open %s\n","showf3.dat");
     if (!(err = wf(SHOWF5DAT))) printf("Couldn't open %s\n","showf5.dat");
     if (!(err = wf(SHOWF7DAT))) printf("Couldn't open %s\n","showf7.dat");
     if (!(err = wf(SHOWF8DAT))) printf("Couldn't open %s\n","showf8.dat");
     if (!(err = wf(SHOWF9DAT))) printf("Couldn't open %s\n","showf9.dat");
     if (!(err = wf(SHOWF11DAT))) printf("Couldn't open %s\n","showf11.dat");
     if (!(err = wf(SHOWF12DAT))) printf("Couldn't open %s\n","showf12.dat");
     if (!(err = wf(SHOWF10DAT))) printf("Couldn't open %s\n","showf10.dat");
     break;                                       /* end EOC case       */

case EOD:
     if (!(err = wf(SHOWF1DAT))) printf("Couldn't open %s\n","showf1.dat");
     if (!(err = wf(SHOWF4DAT))) printf("Couldn't open %s\n","showf4.dat");
     if (!(err = wf(SHOWF6DAT))) printf("Couldn't open %s\n","showf6.dat");
     if (!(err = wf(SHOWF13DAT))) printf("Couldn't open %s\n","statf1.dat");
     break;                                       /* end EOC case       */

case EVMIN:                                       /* start EVMIN case   */
     if (!(err = wf(CURF2DAT))) printf("Couldn't open %s\n","curf2.dat");
     if (!(err = wf(CURF3DAT))) printf("Couldn't open %s\n","curf3.dat");
     if (!(err = wf(CURF1DAT))) printf("Couldn't open %s\n","fault.dat");
     break;                                       /* end EVMIN case     */

} /* end switch */
} /* end write_file */
```

```
/*************************************************************************/
wf(filename)      /* Write output files for Expert System from buffers */
                  /* containing summarized and condensed telemetry.    */
int     filename; /* Files are written at EOC, EOD or Every Minute.    */
{
     FILE   *sfp;          /* Show File# Pointer*/
     int    i,j,err;

switch     (filename)                          /* Write data output file     */
{
case      SHOWF1DAT:                           /* Write showf1.dat, contains */
          err = write_f1("showf1.dat",eod_voltage,1); /* EOD voltages per  */
          return(err);                                /* orbit, per batt   */

case      SHOWF2DAT:                           /* Write showf2.dat, contains */
          err = write_f1("showf2.dat",hc_voltage,2);  /* high_voltages per */
          return(err);                                /* orbit, per battery*/

case      SHOWF3DAT:                           /* Write showf3.dat, contains */
          err = write_f1("showf3.dat",rc_ratio,3);   /* recharge ratio per */
          return(err);                               /* orbit, per battery */

case      SHOWF4DAT:                           /* Write showf4.dat, contains */
          err = write_f2("showf4.dat",cv_eod_hv,cv_eod_lv,cv_eod_av,4);
          return(err);                         /* high, low & avg cell volts */
                                               /* per orbit and per battery  */

case      SHOWF5DAT:                           /* Write showf5.dat, contains*/
          err = write_f2("showf5.dat",cv_hc_hv,cv_hc_lv,cv_hc_av,5);
          return(err);                         /* high, low & avg of 23 cell*/
                                               /* voltages at high charge   */
                                               /* per orbit, per battery    */

case      SHOWF6DAT:                           /* Write showf6.dat, contains*/
          err = write_f3("showf6.dat",cellv_eod,6); /* 23 cell voltages at*/
          return(err);                         /* EOC,per orbit, per battery*/

case      SHOWF7DAT:                           /* Write showf7.dat, contains*/
          err = write_f3("showf7.dat",cellv_hc,7); /* 23 cell voltages at */
                                                   /* high charge per    */
          return(err);                             /* orbit, per battery */
```

```
case     SHOWF8DAT:                              /* Write showf8.dat, contains*/
          if ((sfp = fopen("showf8.dat","w")) >= 0) /* average temp per   */
          {                                        /* batt every 2 mins  */
               fprintf(sfp, "showf(8,[");
               for (i=0; i<6; i++)
               {
                    fprintf(sfp, "[");
                    for (j=0; j<48; j++)
                    {
                         if (j != 47) fprintf(sfp,"%f,",avgt[i][j]);
                         else fprintf(sfp,"%f",avgt[i][j]);
                    }
                    if (i != 5) fprintf(sfp,"],\n");
                    else fprintf(sfp,"]]).");
               }
               fclose(sfp);
               return(SUCCESS);
          }else return(FAIL);

case     SHOWF9DAT:                              /* Write showf9.dat, contains*/
          err = write_fl("showf9.dat",avg_temp,9); /* average temperature */
          return(err);                            /* per orbit, per batt */

case     SHOWF10DAT:                             /* Write showf10.dat, contains*
          if ((sfp = fopen("showf10.dat","w")) >= 0)  /* 23 cell pressures *
          {                                       /* at EOC & EOD for last full *
               fprintf(sfp, "showf(10,[");        /* orbit, per battery         *
               for (i=0; i<6; i++)
               {
                    fprintf(sfp, "[");
                    for (j=0; j<23; j++)
                         fprintf(sfp,"%f,",cp_eod[i][j]);

                    for (j=0; j<23; j++)
                    {
                         if (j != 22) fprintf(sfp,"%f,",cp_eoc[i][j]);
                         else fprintf(sfp,"%f",cp_eoc[i][j]);
                    }
                    if (i != 5) fprintf(sfp, "],\n");
                    else fprintf(sfp, "]]).");
               }
               fclose(sfp);
               return(SUCCESS);
          }else return(FAIL);
```

```
case      SHOWF11DAT:                                  /* Write showf11.dat, contains*
          err = write_fl("showf11.dat",time_tc,11);    /* trickle time per   *
          return(err);                                 /* orbit, per battery *

case      SHOWF12DAT:                                  /* Write showf12.dat, contains*
          if ((sfp = fopen("showf12.dat","w")) >= 0)   /* battery current    *
          {                                            /* during reconditioning at   *
               fprintf(sfp, "showf(12,[");             /* 2 min intervals per battery*
               for (i=0; i<6; i++)
               {
                    fprintf(sfp, "[%d,",rc_orbit[i]);
                    for (j=0; j<48; j++)
                    {
                         if (j != 47) fprintf(sfp,"%f,",bc_drc[i][j]);
                         else fprintf(sfp,"%f",bc_drc[i][j]);
                    }
                    if (i != 5) fprintf(sfp, "],\n");
                    else fprintf(sfp, "]]).");
               }
               fclose(sfp);
               return(SUCCESS);
          }else return(FAIL);

case      SHOWF13DAT:                                  /* Write showl3.dat, contains*
          err = write_fl("showf13.dat",aho,13);        /* AHO per orbit,per battery *
          return(err);

case      CURF1DAT:                                    /* Write curf1.dat, contains  *
          if ((sfp = fopen("curf1.dat","w")) >= 0)     /* orbit number &     *
          {                                            /* reconditioning flags per   *
               fprintf(sfp, "curf(1,[%d,[",hid[1].orbit);   /* battery        *
               for (j=0; j<6; j++)
               {
                    if (j != 5) fprintf(sfp,"%d,",hid[1].batd[j].batrecond);
                    else fprintf(sfp, "%d]]).",hid[1].batd[j].batrecond);
               }
               fclose(sfp);
               return(SUCCESS);
          }else return(FAIL);
```

```
case     CURF2DAT:                                  /* Write curf2.dat, contains*/
         if ((sfp = fopen("curf2.dat","w")) >= 0)     /* AHI,AHO, phase,*/
         {                                          /* 13 SPA currents, average */
                                                    /* temperature for each batt*/
              fprintf(sfp, "curf(2,[%d,%d,\n[",hid[1].phase,hid[1].day_min);

              for (j=0; j<13; j++)
              {
                   if (j != 12) fprintf(sfp,"%f,",hid[1].spac[j]);
                   else fprintf(sfp,"%f",hid[1].spac[j]);
               }
              fprintf(sfp, "],\n[");
              for (j=0; j<3; j++)
              {
                   if (j != 2) fprintf(sfp, "%f,",hid[1].bd[j].busc);
                   else fprintf(sfp, "%f",hid[1].bd[j].busc);
              }
              fprintf(sfp, "],\n[");
              for (j=0; j<6; j++)
              {
                   if (j != 5) fprintf(sfp,"%f,",batt_avgt[j]);
                   else fprintf(sfp,"%f]]).",batt_avgt[j]);
              }
              fclose(sfp);
              return(SUCCESS);
         }else return(FAIL);

case     CURF3DAT:                                   /* Write curf3.dat, contains *
         if ((sfp = fopen("curf3.dat","w")) >= 0)    /* day min, night min *
         {                                           /* 23 cell voltages per batt */
              fprintf(sfp, "curf(3,[");

              for (i=0; i<6; i++)
              {
                   fprintf(sfp, "[");
                   for (j=0; j<23; j++)
                   {
                        if(j != 22)fprintf(sfp,"%f,",hid[1].batd[i].cellv[j]
                        else fprintf(sfp,"%f", hid[1].batd[i].cellv[j]);
                   }
                   if (i != 5) fprintf(sfp, "],\n");
                   else fprintf(sfp, "]]).");
              }
              fclose(sfp);
              return(SUCCESS);
         } else return(FAIL);
```

```
case      FAULTDAT:                                  /* fault.dat has fault   */
          if ((sfp = fopen("fault.dat","w")) >= 0)   /* flag1 = 0 or 1  */
          {                                         /* flag2 = 0 or -1 */
               fprintf(sfp, "fault([%d,%d]).",fault[0],fault[1]);
               fclose(sfp);
               return(SUCCESS);
          }else return(FAIL);

} /* end of switch */
} /* end of wf */
```

```
/*************************************************************************/
write_fl(filename,buffer,number) /* Write output file for Expert System*/
char       *filename;       /* buffers containing summarized telemetry. */
float      buffer[6][12];
int        number;
{
FILE      *file;
int      i,j;

     if ((file = fopen(filename,"w")) >= 0)
     {
          fprintf(file, "showf(%d,[",number);
          for (i=0; i<6; i++)
          {
               fprintf(file, "[");
               for (j=0; j<12; j++)
               {
                    if (j != 11) fprintf(file, "%f,", buffer[i][j]);
                    else fprintf(file, "%f", buffer[i][j]);
               }
               if (i != 5) fprintf(file, "],\n");
               else fprintf(file, "]]).");
          }
          fclose(file);
          return(SUCCESS);
     }else return(FAIL);
} /* end write_fl */
```

```
/*************************************************************************/
write_f2(filename,buffer1,buffer2,buffer3,number)
                  /* Write output files for Expert System from buffers */
char   *filename; /* containing summarized and condensed telemetry.    */
float  buffer1[6][12],buffer2[6][12],buffer3[6][12];
int    number;
{
     FILE    *file;
     int     i,j;

     if ((file = fopen(filename,"w")) >= 0)
     {
          fprintf(file, "showf(%d,[",number);
          for (i=0; i<6; i++)
          {
               fprintf(file, "[");
               for (j=0; j<12; j++)
                    fprintf(file, "%f,",buffer1[i][j]);

               for (j=0; j<12; j++)
                    fprintf(file,"%f,",buffer2[i][j]);

               for (j=0; j<12; j++)
               {
                    if (j != 11) fprintf(file,"%f,",buffer3[i][j]);
                    else fprintf(file, "%f",buffer3[i][j]);
               }
               if (i != 5) fprintf(file, "],\n");
               else fprintf(file, "]]).");
          }
          fclose(file);
          return(SUCCESS);
     }else  return(FAIL);
} /* end write_f2 */
```

```
/*************************************************************************/
write_f3(filename,buffer,number) /* Write output file for Expert System*/
char     *filename;   /* from buffers containing summarized telemetry. */
float    buffer[6][23];
int      number;
{
     FILE     *file;
     int      i,j;

     if ((file = fopen(filename,"w")) >= 0)
     {
          fprintf(file, "showf(%d,[",number);
          for (i=0; i<6; i++)
          {
               fprintf(file, "[");
               for (j=0; j<23; j++)
               {
                    if (j != 22) fprintf(file,"%f,", buffer[i][j]);
                    else fprintf(file,"%f", buffer[i][j]);
               }
               if (i != 5) fprintf(file,"],\n");
               else fprintf(file, "]]).");
          }
          fclose(file);
          return(SUCCESS);
     }else return(FAIL);
} /* end write_f3 */
```

APPENDIX B

CODE AND DOCUMENTATION FOR EXPERT SYSTEM

PROLOG.INI

This is the initialization file for ARITY PROLOG. The directives in this file are run every time the Prolog interpreter is called, by typing "api" from the NICBES directory. Prolog.ini consults all the PROLOG programs needed to run NICBES (start.prg, faultd.prg, utility.prg, showpak.prg, grafpak.prg, status.prg, advice.prg). Any additional routines added to this package should be included in prolog.ini. All the data files necessary for the execution of the Expert System are copied to the NICBES directory. The current data files (curf(N).dat, N = 1 to 3) and fault.datare loaded. begin/0 is called to initiate the Expert System

```
:- ['utility.prg','showpak.prg','grafpak.prg','start.prg','advice.prg',
    'status.prg','faultd.prg'].

% :- shell('copy c:\usr\showf*.dat .'),
%       shell('copy c:\usr\curf*.dat .'),
%       shell('copy c:\usr\fault.dat .').

:-['fault.dat','curf1.dat','curf3.dat','curf2.dat'].

:- begin.
```

Start.prg is the main driver for the NICBES. It calls faultd.prg if a fault flag is set. Otherwise is prints menus to the screen so that the user, by making selections, can determine what portion of the system to view next.

PREDICATES AVAILABLE IN START.PRG

begin/0--calls fault to read the fault flag. It also calls curf(1) to get the orbit number and reconditioning flags, which are both asserted to the data base for use by other routines. begin/1 then calls eval_flag/1. begin/0 is called by prolog.ini.

eval_flag(Flag)--if Flag is 1, control is passed to faultd.prg for fault diagnosis. When finished, the user is asked whether more information is desired. The user's response is validated by check1/1 and then continue/1 is called. If Flag is 0, eval_flag/1 consults all the show files to load the data as facts. complete/1 is then called to insure that enough values are present in the data set for accurate analysis. If there are too many missing data points the user is only allowed to view the graphics portion of the Expert System. sequence/1 is called next to prepare the horizontal axis for plots which will be drawn later. write_message/1 then brings up the Main Menu. eval_flag/1 called by begin/0.

continue(yes/no)--If yes, eval_flag(0) is called. If no, all data files are deleted from the NICBES directory and PROLOG is halted. The user is returned to DOS.

write_message(N)--brings the Main Menu to the screen and waits for user input which is checked for validity by check2/2. N is a flag which determines the allowable selections by the user. write_message/1 operates in a repeat-fail loop so that the user can always come back to the Main Menu. There are 4 selections in the menu:

```
NICBES MAIN MENU
          1.  PLOTS AND GRAPHS
          2.  BATTERY STATUS
          3.  ADVICE ON RECONDITIONING, WORKLOAD AND CHARGE
          4.  Quit NICBES
ENTER YOUR SELECTION (e.g. 1<CR> ):
```

write_message/1 is called by eval_flag(0). It calls battery/1.

battery(Choice)--Choice (1 - 4) is the item selection from the Main Menu. battery(4) calls continue(no) to halt the Expert System. battery/1 asks the user to select a battery. The user's response is validated by check/2. execute/2 is then called to execute the Main Menu Choice for

the chosen battery. battery/1 is in a repeat-fail loop so that the the user can make multiple battery selections.

execute(Choice,Bat)--is called by write_message/1 to pass control to the Choice selected in the Main Menu. There are 3 cases.

execute(1,Bat)--Plots and Graphs. A Graphics Menu is written on the screen with 12 plots per battery. The user may also make another Battery Selection or Quit to the Main Menu. User's choice is checked for validity by calling check/2. After the graph has been drawn on the screen the user is asked to enter '<CR>' when ready to continue. execute/2 is in a repeat-fail loop so that the Graphics Menu will always return until the user opts to return to the Main MEnu. show_view/3 (showpak.prg) is called to generate the plots.

GRAPHICS MENU FOR BATTERY N

1--Battery Voltage at EOD for last 12 orbits
2--Battery Voltage at high in-charge for last 12 orbits
3--Recharge ratio for last 12 orbits
4--Cell Voltages at EOD; high, low, average for last 12 orbits
5--Cell Voltages at high in-charge; high, low, avg, last 12 orbits
6--Cell Voltages at EOD for latest orbit
7--Cell Voltages at high in-charge for latest orbit
8--Average Battery Temperature for latest orbit, each 2 min
9--Average Battery Temperature for last 12 orbits
10-Cell Pressures at EOC and EOD for latest orbit
11-Time on Trickle Charge for last 12 orbits
12-Battery Current during reconditioning, 1 orbit, each 2 min
13-Quit for Another Battery Selection
14-Quit to Main Menu
Enter choice (e.g. 5<CR>):

execute(2,Bat)--activates the Status analysis portion. A header is written to the screen, STATUS FOR BATTERY N, LATEST ORBIT NN. Then battery_status/1 is called to analyze Battery Status after which the user can opt to make another Battery selection and return to Status or Quit to the Main Menu.

execute(3,Bat)--An Advice Menu of five items is displayed to the user. User's response is checked for validity by check/2. more/2 is called to activate the Advice portion of the Expert System. execute/2 is in a repeat-fail loop so that the Advice Menu will always return until the user opts to return to the Main Menu.

BATTERY ADVICE MENU FOR BATTERY N

1. RECONDITION BATTERY?
2. CHANGE CHARGING REGIME?
3. CHANGE WORKLOAD?
4. QUIT FOR ANOTHER BATTERY SELECTION
5. QUIT TO MAIN MENU

ENTER CHOICE (e.g. 1<CR>):

more(Choice,Bat)--writes a header to the screen, ADVICE FOR BATTERY N, LATEST ORBIT NN. advice/2 (advice.prg) is then called. more/2 is called by execute(3,_).

```
%  start.prg is the main control unit for the Expert System.  It calls
%  fault_diagnosis if fault flag is set to 1.  Main Menu and supporting
%  menus are drawn on the screen.  Control is passed to other portions of
%  the Expert System depending on the user's selections.
%  See start.doc for further documentation.

begin:-
      call(fault([Flag,_])),
      call(curf(1,[Orbit,Recond])),
      asserta(orbit(Orbit)),
      asserta(recond(Recond)),
      eval_flag(Flag).

eval_flag(1) :-
      cls,nl,
      write($**************************************************$),
      nl,write($NICBES FAULT DIAGNOSIS FOR ORBIT $),
      call(orbit(Orb)), write(Orb),nl,
      fault_diag,nl,
      write($**************************************************$),
      nl,
      repeat,
      write($  Do you want more information (yes or no)?  $),
      reader(Ans),
      check1(Ans),
      continue(Ans).

continue(yes):-
      eval_flag(0).

continue(no):-
%     shell('del showf*.dat'),
%     shell('del curf*.dat'),
%     shell('del fault.dat'),
      halt.

eval_flag(Flag) :-
      ['showf1.dat','showf2.dat','showf3.dat','showf4.dat','showf5.dat',
      'showf6.dat','showf7.dat','showf8.dat','showf9.dat','showf10.dat',
      'showf11.dat','showf12.dat','showf13.dat'], .
      complete(N),
      call(orbit(Orbit)),
      sequence(Orbit),
      write_message(N).
```

```
write_message(N) :-
      cls,nl,nl,
      repeat,
      write($      NICKEL CADIUM BATTERY EXPERT SYSTEM$),nl,
      write($      ----------------------------------------$),nl,nl,nl,
      write($      NICBES MAIN MENU$),nl,nl,
      write($                  1  PLOTS AND GRAPHS $),nl,
      write($                  2  BATTERY STATUS $),nl,
      write($                  3  ADVICE ON RECONDITIONING, WORKLOAD OR CHARGE$),
      nl,
      write($                  4  QUIT NICBES$),nl,nl,
      write($  ENTER YOUR SELECTION (e.g. 1<CR> ):  $),
      reader(Choice),
      check2(Choice,N),
      battery(Choice),
      fail.

battery(4) :-
      continue(no).

battery(Choice) :-
      cls,nl,nl,nl,
      repeat,
      nl,write($   NICKEL CADIUM BATTERY EXPERT SYSTEM  $),
      nl,write($   ---------------------------------- $),
      nl,nl,
      write($  ENTER BATTERY SELECTION 1 - 6 (e.g.  2<CR> ):  $),
      reader(Bat),nl,nl,
      check(Bat,6),
      execute(Choice,Bat),
      cls,nl,nl,
      !,fail.

execute(2,Bat) :-
      write($**************************************************$),nl,
      write($  STATUS FOR BATTERY $),write(Bat),
      call(orbit(Orbit)),
      write($, LATEST ORBIT $),write(Orbit),nl,nl,
      battery_status(Bat),
      nl,write($**************************************************$),
      nl,nl,
      write($      1  QUIT FOR ANOTHER BATTERY SELECTION $),nl,
      write($      2  QUIT TO MAIN MENU  $),nl,
      write($  ENTER CHOICE (e.g.  1<CR>):  $),
      reader(Choice),
      check(Choice,2),
      ((Choice == 1, !, fail);(Choice == 2, cls, nl, nl)).
```

```
execute(1,Bat) :-
      call(orbit(Orbit)),
      cls,nl,nl,
      repeat,
      write($GRAPHICS MENU FOR BATTERY $),write(Bat),nl,nl,
      write($    1  Battery Voltage at EOD for last 12 orbits $), nl,
      write($    2  Battery Voltage at high in-charge for last 12 orbits $), nl,
      write($    3  Recharge ratio for last 12 orbits$), nl,
      write($    4  Cell Voltages at EOD; high, low, average for last 12 orbits$)
      write($    5  Cell Voltages at high charge; high,low avg, last 12 orbits$),
      nl,
      write($    6  Cell Voltages at EOD for latest orbit $), nl,
      write($    7  Cell Voltages at high in-charge for latest orbit $), nl,
      write($    8  Average Battery Temperature for latest orbit, each 2 min$),nl
      write($    9  Average Battery Temperature for last 12 orbits$),nl,
      write($   10  Cell Pressures EOC and EOD for last orbit$),nl,
      write($   11  Time on Trickle Charge for last 12 orbits$), nl,
      write($   12  Battery Current during reconditioning, 1 orbit-each 2 min $),
      nl,
      write($   13  Quit for Another Battery Selection$),nl,
      write($   14  Quit to Main Menu $),nl,
      write($ENTER CHOICE (e.g.  1<CR> ):  $),
      reader(Choice),
      check(Choice,14),
      ((Choice == 13,!,fail);
      ((Choice == 14,cls,nl,nl);
      (show_view(Choice,Bat,Orbit),
      tmove(0,0),
      write($ENTER  <CR>  WHEN READY TO CONTINUE:  $),
      reader(Ans),
%     gc(full),
      cls,fail))).

execute(3,Bat) :-
      repeat,
      write($  BATTERY ADVICE MENU FOR BATTERY $),write(Bat),
      nl,nl,
      write($                  1  RECONDITION BATTERY?  $),nl,
      write($                  2  CHANGE CHARGING REGIME?  $),nl,
      write($                  3  CHANGE WORKLOAD?  $),nl,
      write($                  4  QUIT FOR ANOTHER BATTERY SELECTION  $),nl,
      write($                  5  QUIT TO MAIN MENU  $),nl,nl,
      write($  ENTER CHOICE  (e.g.  1<CR> ):  $),
      reader(Choice),
      check(Choice,5),
      ((Choice == 4,!,fail);
      ((Choice == 5,cls,nl,nl);
      (more(Choice,Bat),fail))).
```

```
more(Choice,Bat) :-
      write($**************************************************$),nl,
      call(orbit(Orbit)),
      write($ADVICE FOR BATTERY $),write(Bat),
      write($, LATEST ORBIT $),write(Orbit),nl,nl,
      advice(Bat,Choice),
      write($**************************************************$),
      nl,nl,
      write($  TO CONTINUE ENTER  <CR> $),
      reader(Ans),cls,!.
```

Faultd.prg is the fault diagnosis section of NICBES. It uses the current data contained in curf1.dat, curf2.dat, and curf3.dat to diagnose alarms to the system. Faultd.prg is automatically invoked by the system when the C language data handler detects an anomaly. It loads values into the database, diagnoses the problem, and cleans up the database, then quits. It is not accessible by the user. If more than one fault exists, it will diagnose them all.

ALARMS FOR NICBES SYSTEM:

More than 18 faults trigger the suggestion to "Check the control computer." This is a means to quantify the statement "lots of crazy data coming out means the control computer is faulty."

The data handler monitors the telemetry as it comes in, and looks for values that indicate a serious malfunction of the system, for which the DEC LSI-11 has sent an alarm to the engineer. The alarms are in five categories:

1. Power supplies

Current from any SPA less than 5 A. during first five minutes of the day is a failure of power supply or control circuitry. Current greater than 8 A. for 1-SPAs (1,3,5,7,9,11) or 16 A. for 2-SPAs (2,4,6,8,10,12,13) is a failure in control circuitry.

a. current < 5 A during first 5 minutes of charge period
   --check the 13 SPA currents (SPA I), phase is 1, day min. <= 5.

b. current >= 8 A for 1-SPAs, >= 16 A for 2-SPAs
   --check the 13 SPA currents.

c. current > 5 A during discharge period
   --check the 13 SPA currents, phase is 0.

2. Batteries

Cell voltage for any cell less than 0 means a failure in BPRC, and voltage of 0 means a short to a cell. Cell voltage for any cell greater than 1.55V. is a failure in control limit circuitry.

a. Cell voltage <= 0 for any cell in any battery
   --check cell voltages

b. Cell voltage > 1.55 for any cell in any battery

--check cell voltages

3. Load banks

If the sum of the current on busses A, B and C (V1, V2, V3 in code) is greater than 99 A., bus overload has occurred, a failure of the control circuitry or load banks. If any of the bus currents is less than 5 A. during discharge period, a failure of control circuitry has occurred.

a. Sum of three bus currents > 99 A
   --sum the three bus currents (Bus I).

b. Load < 5 A on any single bus during discharge period
   --check the three bus currents when phase = 0.

4. Temperature

If 1 battery has high temperature (> 25C) during charge cycle, thermal runaway is suggested. If 2 or more have high temperature, it is suggested that the chamber be checked for cooling malfunction. If 1 or more batteries are cold (< -10C), it is suggested that the chamber be checked for failure of coolant control (failed ON).

Average of the six temperature sensors > 25 degrees or < -10 degrees
   --take the average of the six temp sensors.

5. Communication

More than two telemetry runs missed mean either a system shutdown or a communication failure to the PC and will trigger an alarm diagnosis by NICBES. NICBES resets the fault.dat flag file to 0. after an alarm diagnosis, so it is ready to start again when the system is restarted.

No data coming into PC.

CURRENT DATA FILES:

curf(2,[Phase,Day_min,Curlist,Vlist,Templist]).
where Curlist contains 13 SPA Currents
      Vlist contains 3 Bus Currents
      Templist contains Average Temperature for 6 Batteries

curf(3,[Voltlist]).
where Voltlist contains 6 sublists, one for each battery,
      with 23 Cell Voltages each.

fault([fault flag, Type]).

THE PREDICATES AVAILABLE IN FAULTD.PRG

fault_diag--uses the current data files loaded in the database by prolog.ini to find the faults which cause alarms to the testbed system. Five classesof faults are diagnosed by calls to comm_fail/0, power_cir_fail/0, loadbank_fail/0, chamber_fail/0, and cell_fail/0. A message is written to the screen as each of the above are checked. If any of these five predicates discover a fault, they write a message to the screen. After diagnosis is done, diagnose/0 checks the counter 20. If more than 18 fault lines have been written in fault diagnosis, it suggests that the control computer be checked for sending bad data. It would be unlikely for that many faults to occur otherwise. diagnose/0 is called automatically from eval_flag(1) (start.prg).

comm_fail--checks fault([_,Type]). The Data-Handler will set this to -1 if it detects a failure in communication, 3 missed telemetry runs. comm_fail/0 is called by diagnose/0. Fault diagnosis can still analyze last set of current data files. This may however be misleading so handle with caution. If no problem found, a message will be written to the screen so stating.

chamber_fail--checks battery temperatures from Templist. If only one battery is hot (> 25C), it suggests checking for thermal runaway. If more than one battery is hot, it suggests checking coolant control. If any battery is cold (< -10C), it suggests checking coolant control failed on. chamber_fail/0 will always succeed, so that other faults can be diagnosed as well. If no problems found, a message will be written to the screen so stating. chamber_fail/0 calls check_temp/2 . It is called by fault_diag/0.

check_temp(Templist,Phase)--calls check_t to recursively check the list of temperatures. Counters 1 and 2 are used to store the number of high or low values found, respectively. Counter 3 is used to keep track of which batteries temperatures are being checked. check1_t/3 is called to check highs and lows. Diagnostic messages are printed. check_temp/2 is called by chamber_fail/0.

check_t([H|T])--checks the head of the list (H) for missing data (-9999), or temperature out of range, prints a message if either condition holds, then calls itself on the tail of the list (T). Counter 3 keeps track of which battery is being checked. Counter 1 keeps track of how many high values, and counter 2 counts low values.

check1_t(Highs,Lows,Phase)--checks number of Highs and Lows. Diagnostic messages are written to the screen.

loadbank_fail--uses Phase and Vlist from curf(2,_). check_bus/2 and check1_bus/2 are called to check for problems in the load bank. find_sum/2 is called to get the sum of the 3 bus voltages. loadbank_fail/0 always succeeds, so that other problems can also be diagnosed. If no problems found, a message will be written to

the screen so stating. load_bank/0 is called from fault_diag/0.

check_bus(Vlist,Phase)--checks bus values for missing data (-9999). If any bus has less than 5 A during discharge phase, failure of load bank or control circuitry is diagnosed. check_bus/4 is called by loadbank_fail/0.

checkl_bus(Sum,Phase)--if the sum of the bus currents is over 99A, failure of load bank or control circuitry is diagnosed. Called by loadbank_fail/0.

power_cir_fail--uses Day_min, Phase and Curlist from curf(2,_), then calls check_cur/3 to diagnose failures in SPAs. Counter 3 is used to keep track of which bus is being checked. power_cir_fail/0 always succeeds, so other faults can also be diagnosed. If no problems found, a message will be written to the screen so stating. power_cir_fail is called by fault_diag/0.

check_cur([H|T],Phase,M)--checks data from the list of SPA currents (Curlist) recursively, checking an item, then calling itself on the tail of the list. It checks for missing data (-9999), then checks for malfunctions in SPAs. Possible malfunctions are current less than 5 A in first five minutes of charge period, current from any SPA in discharge period, current of greater than 8 A from 1-SPAs (1,3,5,7,9,11), or current of greater than 16 A from 2-SPAs (2,4,6,8,10,12,13). check_cur/3 writes diagnostic messages to the screen. It is called by power_cir_fail/0.

cell_fail--uses Voltlist from curf(3,_). Calls check_volt/1 to check each battery cell for problems. If no problems found, a message will be written to the screen so stating. cell_fail/0 is called by fault_diag/0.

check_volt(Voltlist)--sets counter 3 to keep track of which cell is being checked. It recurses through the 6 lists of cell voltages, one for each battery. The recursive function read_volt/2 is called to check the voltages. check_volt/2 is called from cell_fail/0.

read_volt([H|T],N)--checks the head of the list (H) for missing data (-9999), voltage of 0 (diagnoses hard short), negative voltage (diagnoses BPRC failure), and voltage over 1.55 (diagnoses overcharge). read_volt/2 then calls itself on the tail of the list (T). Counter 3 is used to keep track of which cell is being checked, for use in the failure messages. read_volt/2 is called from check_volt/1.

```
% Faultd--fault diagnosis section of NICBES.  Checks five fault
% conditions.  See fault.doc for further documentation.

fault_diag :-
     ctr_set(20,0),
     nl,write($  CHECKING COMMUNICATIONS:$),
     comm_fail,
     nl, write($  CHECKING SPA CURRENTS:$),
     power_cir_fail,
     nl, write($  CHECKING BUS CURRENTS:$),
     loadbank_fail,
     nl, write($  CHECKING BATTERY TEMPERATURES:$),
     chamber_fail,
     nl, write($  CHECKING CELL VOLTAGES:$),
     cell_fail,
     ctr_is(20,Faults),
     ifthen(Faults == 0,
     (nl,write($**No cause found for alarm--check alarm     system.$))),
     ifthen(Faults > 18,
     (nl,write($-----Many failures--check control computer-----$))).

comm_fail :-
     call(fault([_,-1])),nl,
     write($**Communication failure--receiving no data from system.$),nl,
     write($  Fault diagnosis is limited to last set of current data files.$),
     ctr_inc(20,_).

comm_fail :- write($  No Communications Problems Found!  $).

chamber_fail :-
     call(curf(2,[Phase,_,_,_,Templist])),
     check_temp(Templist,Phase).

chamber_fail :-
     (retract(flag) ; write($  No Chamber Problems Found!  $)).

check_temp(Templist,Phase) :-
     ctr_set(1,0),ctr_set(2,0),ctr_set(3,1),
     check_t(Templist),
     ctr_is(1,Highs),
     ctr_is(2,Lows),
     check1_t(Highs,Lows,Phase).
```

```
check_t([]).

check_t([-9999|T]) :-
      ctr_inc(3,N),
      nl,write($ Temp data item missing on battery $),write(N),
      check_t(T).

check_t([H|T]) :-
      H > 25,
      ctr_inc(3,N),
      ctr_inc(1,_),
      nl,write($**Battery $), write(N),
      write($ is overheated--$),write(H),write($ degrees.$),
      one_assert(flag),
      ctr_inc(20,_),
      check_t(T).

check_t([H|T]) :-
      H < -10,
      ctr_inc(3,N),
      ctr_inc(2,_),
      nl,write($**Battery $),write(N),
      write($ is too cold at $),write(H),write($ degrees.$),
      one_assert(flag),
      ctr_inc(20,_),
      check_t(T).

check_t([H|T]) :-
      ctr_inc(3,_),
      check_t(T).

check1_t(_,Lows,_) :-
      Lows > 0,
      nl,write($**Chamber too cold--check coolant control.$),
      ctr_inc(20,_),
      !,fail.

check1_t(1,_,1) :-
      nl,write($**Check for thermal runaway.$),
      ctr_inc(20,_).

check1_t(Highs,_,_) :-
      Highs > 2,
      nl,write($**Overheated chamber--check coolant control.$),
      ctr_inc(20,_).
```

```
loadbank_fail :-
      call(curf(2,[Phase,_,_,Vlist,_])),
      ctr_set(3,1),
      check_bus(Vlist,Phase),
      find_sum(Vlist,Sum),
      check1_bus(Sum,Phase).

loadbank_fail :-
      (retract(flag); write($  No Loadbank Problems Found!$)).

check_bus([],_).

check_bus([-9999|T],Phase) :-
      ctr_is(3,N),
      nl,write($Missing data on bus $),write(N),
      ctr_inc(3,_),
      check_bus(T,Phase).

check_bus([H|T],Phase) :-
      Phase == 0,
      H < 5,
      ctr_is(3,N),
      nl,write($**Load on Bus $),
      write(N),write($ 1 < 5 amps--load bank or control failure$),
      ctr_inc(3,_),ctr_inc(20,_),
      one_assert(flag),
      check_bus(T,Phase).

check_bus([H|T],Phase) :-
      ctr_inc(3,_),
      check_bus(T,Phase).

check1_bus(Sum,Phase) :-
      Sum > 99,
      nl,write($**Bus currents > 99 amps--failure in load banks or control
           circuitry.$),
      ctr_inc(20,_).

power_cir_fail :-
      call(curf(2,[Phase,Daymin,Curlist,_,_])),
      ctr_set(3,1),
      check_cur(Curlist,Phase,Daymin).

power_cir_fail:- write($  No SPA Current Problems Found!$).
```

```
check_cur([],_,_).

check_cur([-9999|T],Phase,Daymin) :-
      ctr_inc(3,L),
      nl,write($Missing data from SPA $),write(L),nl,
      check_cur(T,Phase,Daymin).

check_cur([H|T],Phase,Daymin) :-
      ctr_is(3,L),
      L =\= 13,
      Phase == 1,
      Daymin < 7,
      H < 5,
      H > -50,
      ctr_inc(3,_),
      nl,write($**Current low from SPA-$), write(L),
      write($ in first 5 minutes of day--$),nl,
      write($  failure of power supply or control.$),
      ctr_inc(20,_),
      check_cur(T,Phase,Daymin).

check_cur([H|T],Phase,Daymin) :-
      Phase == 0,
      H > 5,
      ctr_inc(3,L),
      nl,write($**Current from SPA-$), write(L),
      write($ during night--control circuitry failure$),
      ctr_inc(20,_),
      check_cur(T,Phase,Daymin).

check_cur([H|T],Phase,Daymin) :-
      ctr_is(3,L),
      L =\= 13,
      K is L mod 2,
      K == 1,
      H >= 8,
      ctr_inc(3,_),
      nl,write($**Current from SPA-$), write(L), write($ >= 8 amps.$), nl,
      write($  Check for failure in control limit circuitry.$),
      ctr_inc(20,_),
      check_cur(T,Phase,Daymin).
```

```
check_cur([H|T],Phase,Daymin) :-
      ctr_is(3,L),
      K is L mod 2,
      (K == 0;L == 13),
      H >= 16,
      ctr_inc(3,_),
      nl,write($**Current from SPA-$),write(L),write($ >= 16 amps.$),nl,
      write($  Check current limit circuitry.$),
      ctr_inc(20,_),
      check_cur(T,Phase,Daymin).

check_cur([H|T],Phase,Daymin) :-
      ctr_inc(3,_),
      check_cur(T,Phase,Daymin).

cell_fail :-
      call(curf(3,Voltlist)),
      ctr_set(1,1),
      check_volt(Voltlist).

cell_fail :-
      (retract(flag); write($ No Cell Failures Found!$)).

check_volt([]).

check_volt([H|T]) :-
      ctr_inc(1,N),
      ctr_set(3,1),
      read_volt(H,N),
      check_volt(T).

read_volt([],_).

read_volt([-9999|T],N) :-
      ctr_inc(3,L),
      nl,write($Missing data on cell $),write(L),
      write($ of battery $),write(N),
      read_volt(T,N).

read_volt([0|T],N) :-
      ctr_inc(3,L),
      nl,write($**Suspect hard short in cell $),write(L),
      write($ of battery $),write(N),
      ctr_inc(20,_),
      one_assert(flag),
      read_volt(T,N).
```

```
read_volt([H|T],N) :-
      H < 0,
      ctr_inc(3,L),
      nl,write($**BPRC failed in cell $),write(L),
      write($ of battery $),write(N),
      ctr_inc(20,_),
      one_assert(flag),
      read_volt(T,N).

read_volt([H|T],N) :-
      H > 1.55,
      ctr_inc(3,L),
      nl,write($**Voltage too high in cell $),write(L),
      write($ of battery $),write(N),nl,
      write($  check overcharge or too high a charge rate.$),
      ctr_inc(20,_),
      one_assert(flag),
      read_volt(T,N).

read_volt([H|T],N) :-
      ctr_inc(3,L),
      read_volt(T,N).
```

Status.prg has the STATUS portion of NICBES. Selecting Battery Status from the Main Menu for battery n (n is 1 through 6) will trigger the checking of battery n for reconditioning, temperature, workload, charging scheme, and divergence. Messages will be printed on the screen. The STATUS section uses averages of data from some of the showf(n).dats in its analysis. If nothing is wrong with the battery, a message is written to this effect. Problems with the battery often produce messages to use "Plots and Graphs" or "Advice" for more information.

The following conditions are examined:

Condition	Status--
Temperature average over last orbit < 0, and over last 12 orbits < 0	Cold
Temperature avg. over last orbit > 11, and over last 12 orbits > 10	Hot or possible overcharging
Temperature avg. over last orbit > 11, and over last 12 orbits < 10	Possible overcharging
AHO average over last 12 orbits < 3 (3.0 Ampere-hours), and avg. EOD voltage < 27	Underwork
AHO average over last 12 orbits > 14 (14.0 Ampere-hours)	Overwork
Recharge ratio average < 1.020 and avg. high-charge voltage < 32.5V	Insufficient charge
High-charge voltage > 33.8	Possible overcharge
Divergence average > .8V between high and low cells	High divergence
High cell minus average greater than (avg. minus low cells) + 1	Too many cells to low values

---

PREDICATES AVAILABLE IN STATUS.PRG

battery_status(Bat)--First, battery_status/1 checks for reconditioning. If the battery is being reconditioned, status analysis stops at that point, since data will be misleading. Otherwise, four procedures are called, temp_status/1, work_status/1, charge_status/1, and div_status/1. Each of these procedures

writes messages to the screen and increments counter 15 if problems are found. If counter 15 is still zero, a message is written to the screen that the battery seems healthy. battery_status/1 is called from execute(2,Bat) (start.prg).

temp_status(Bat)--showf(8,_) (temperature over last orbit) and showf(9,_) (average temperature over last 12 orbits) are used in analysis. get_data/5 is called to get the appropriate battery data and then to find the its average. Finally, check_temp/3 is called to analyze the data. temp_status/1 will always succeed, even if no problem is found, so that status analysis can continue. If no problem was found a message will be printed to the screen. temp_status/1 is called from battery_status/1.

work_status(Bat)--showf(13,_) (AHO for last 12 orbits) and showf(1,_) (EOD battery voltage over last 12 orbits) is used to analyze workload status. get_data/5 is called to get the appropriate battery data and then to find the its average. Analysis is done by calling check_work/3. find_avg/2 is called to find averages. work_status/1 will always succeed, so that status analysis can continue. If no problem was found a message will be printed to the screen. It is called from battery_status/1.

charge_status(Bat)--showf(3,_) (recharge ratio) and showf(2,_) (high cell voltage during charge) is used to analyze charge status. get_data/5 is called to get the appropriate battery data and then to find the its average. Analysis is done by calling check_charge/3. charge_status/1 will always succeed. If no problem was found a message will be printed to the screen. It is called from battery_status/1.

get_data(Bat,N1,N2,Avg1,Avg2)--N1 and N2 reference the show files, Bat is the selected battery. get_list/3 is called to get the data list for the appropriate show file and battery. find_avg/2 is then called to calculate the average for each data list. Cut is used to prevent backtracking. get_data/5 is called by temp_status/1, work_status/1, charge_status/1.

div_status(Bat)--showf(4,_) (EOD divergence) is used to analyze divergence. get_list/3 is used to get the appropriate battery data, find_div/5 is called to find highs, lows and averages, then eval_div is used to analyze the divergence. div_status/1 will always succeed. If no problem was found a message will be printed to the screen. It is called from battery_status/1.

check_recond(List)--checks the single value passed it by battery_status/1. If this value is 1, the battery is in reconditioning, and a message is printed on the screen. No further status analysis is done.

check_temp(Oavg,Avg12,Bat)--If average temperature in last orbit and average temperature over last 12 orbits are both below zero, a message that the battery is cold is given. If average temperature

over the last orbit is over 11, and average over last 12 orbits is over 10, a message is given that the battery is hot. If average over last orbit is greater than 11, and average over last 12 orbits is less than 10, a message is given to check chamber or check for overcharging. check_temp is called by temp_status/1.

check_work(Bat,Ahoavg,Eodv)--analyzes the battery for overwork or underwork. If the average AHO is less than 3 ampere-hours, and EOD voltage is less than 27 V, a message that the battery may have memory effect is given. If the average AHO is greater than 14 ampere-hours, a message is given that the battery is overworked. check_work/3 is called from work_status/1.

check_charge(Rr,Cavg,Bat)--analyzes the battery for overcharging or undercharging. If recharge ratio is greater than 1.020, and high battery voltage during charge is less than 32.5, a message is given that the charging scheme may be insufficient. If in-charge high voltage is greater than 34 volts, a message is given that possible dangerous overcharging may be occurring. In-charge high voltage average greater than 33.6 volts triggers a message to check for overcharging. check_charge/3 is called from charge_status/3.

find_div(List,Avg,Div,Highs,Lows)--Given a List taken from showf(4,_) or showf(5,_), find_div/5 calls break_list/4 to break the list into a list of high values, a list of low values, and a list of average values for cell voltages. find_avg/2 is then called on each of these lists to find the average high reading (Highs), low reading (Lows) and average reading (Avg) over the 12 orbits. Div is instantiated to Highs minus Lows. find_div/5 is called by div_status/1.

eval_div(Div,Avg,Highs,Lows,Bat)--evaluates two measures of divergence, using variables passed out from find_div. eval_div/5 writes a message that divergence is high if Div is greater than 0.8. If Avg is closer to Lows than to Highs by a noticeable margin (1), a message is given that too many cells are migrating to low values. eval_div is called by div_status/1.

```
% This section has status portion of NICBES
% status(Bat) examines the battery status from five viewpoints
% See Status.doc for documentation


battery_status(Bat) :-
     call(recond(Record)),
     find_nth(Record,1,Bat,List),
     write($CHECKING RECONDITIONING:$),
     check_recond(List).

battery_status(Bat) :-
     write($     Battery is not being reconditioned! $),
     ctr_set(15,0),
     temp_status(Bat),
     work_status(Bat),
     charge_status(Bat),
     div_status(Bat),
     ctr_is(15,A),
     ifthen((A < 1),
     (nl,write($**Battery seems healthy; no obvious problems.$))).

temp_status(Bat) :-
     nl,write($CHECKING TEMPERATURE:$),
     get_data(Bat,8,9,Avg,Avg1),
     check_temp(Avg,Avg1,Bat).

temp_status(Bat) :- write($     No Battery Temperature Problems!  $),nl.

work_status(Bat) :-
     nl,write($CHECKING WORKLOAD:$),
     get_data(Bat,13,1,Avg,Avg1),
     check_work(Bat,Avg,Avg1).

work_status(Bat) :- write($     No Battery Workload Problems!  $),nl.

charge_status(Bat) :-
     nl,write($CHECKING CHARGE:$),
     get_data(Bat,3,2,Avg,Avg1),
     check_charge(Avg,Avg1,Bat).

charge_status(Bat) :- write($     No Battery Charging Problems!  $),nl.

get_data(Bat,N1,N2,Avg1,Avg2) :-
     get_list(Bat,N1,List),
     find_avg(List,Avg1),
     get_list(Bat,N2,List1),
     find_avg(List1,Avg2),!.
```

```
div_status(Bat) :-
     write($CHECKING DIVERGENCE:$),
     get_list(Bat,4,List),
     find_div(List,Avg,Div,Highs,Lows),
     eval_div(Div,Avg,Highs,Lows,Bat).

div_status(Bat) :- write($     No Battery Divergence Problems!  $),nl.

check_recond([1]) :-
     nl,
     write($**Battery is in reconditioning.$),nl,
     write($**No further Status, as analysis would be misleading.**$).

check_temp(Oavg,Avg12,Bat) :-
     Oavg < 0,
     Avg12 < 0,
     ctr_inc(15,_),nl,
     write($**Battery is cold--check chamber$),nl,
     write($  Select Plots and graphs from Main Menu for Battery $),
     write(Bat),nl,
     write($    with choice 8 from Graphics Menu $),nl.

check_temp(Oavg,Avg12,Bat) :-
     Oavg > 11, Avg12 > 10,
     ctr_inc(15,_),nl,
     write($**Battery is hot--check chamber or overcharging$),nl,
     write($  Select Plots and Graphs from Main Menu for Battery $),
     write(Bat),nl,
     write($    with choice 8 from Graphics Menu to see temp,$),nl,
     write($    and with choice 2 from Graphics Menu for overcharging.$),
     nl.

check_temp(Oavg,Avg12,Bat) :-
     Oavg > 11, Avg12 < 10,
     ctr_inc(15,_),nl,
     write($**Battery is hot this cycle--check for overcharge$),nl,
     write($  Select Plots and Graphs from Main Menu for Battery $),
     write(Bat),nl,
     write($    with choice 2 from Graphics Menu for overcharging$),nl.

check_work(Bat,Ahoavg,Eodv) :-
     Ahoavg < 3,
     Eodv < 27,
     ctr_inc(15,_),nl,
     write($**Battery may have memory effect.$),nl,
     write($  Select Plots and Menus from Main Menu for Battery $),
     write(Bat),nl,
     write($    with choice 1 from Graphics Menu for plot of EOD voltage.$),nl,
     write($  Select Advice from Main Menu for Battery $),
     write(Bat),nl,
     write($    with choice from Advice Menu for reconditioning.$),nl.
```

```
check_work(Bat,Ahoavg,Eodv) :-
      Ahoavg > 14,
      ctr_inc(15,_),nl,
      write($**Battery may be overworked. $),nl,
      write($  Select Advice from Main Menu for Battery $),
      write(Bat),nl,
      write($    with choice from Advice Menu for workload.$),nl.

check_charge(Rr,Cavg,Bat) :-
      Rr < 1.020,
      Cavg < 32.5,
      ctr_inc(15,_),nl,
      write($**Battery charging scheme may be insufficient. $),nl,
      write($  Select Advice from Main Menu for Battery $),
      write(Bat),nl,
      write($    with choice from Advice Menu for workload.$),nl.

check_charge(_,Cavg,Bat) :-
      Cavg > 34,
      ctr_inc(15,_),nl,
      write($**Possible dangerous overcharging.**$),nl,
      write($  Select Advice from Main Menu for Battery $),
      write(Bat),nl,
      write($    with choice from Advice Menu for charging.$),nl.

check_charge(_,Cavg,Bat) :-
      Cavg > 33.6,
      ctr_inc(15,_),nl,
      write($**Check for overcharging.**$),nl,
      write($  Select Advice from Main Menu for Battery $),
      write(Bat),nl,
      write($    with choice from Advice Menu for charging.$),nl.

find_div(List,Avg,Div,Highs,Lows) :-
      break_list(List,List1,List2,12),
      break_list(List2,List3,List4,12),
      find_avg(List1,Highs),
      find_avg(List3,Lows),
      find_avg(List4,Avg),
      Div is Highs - Lows,!.

eval_div(Div,Avg,Highs,Lows,Bat) :-
      A is Highs - Avg,
      B is Avg - Lows,
      C is B + 1,
      C < A,
      nl,write($**Too many cells migrating to low values.**$),
      ctr_inc(15,_),fail.
```

```
eval_div(Div,Avg,Highs,Lows,Bat) :-
     Div > 0.8,
     nl,write($**Divergence is high, may need reconditioning.**$),
     nl,write($  Select Advice from Main Menu for Battery $),
     write(Bat),nl,
     write($     with choice from Advice Menu for reconditioning.$),
     ctr_inc(15,_).
```

Advice.prg contains the ADVICE section of NICBES which goes into further detail on three subjects: whether a battery needs reconditioning, changes in charging scheme, or changes in workload. Select Advice from the Main Menu for battery n brings up a menu of three choices for the user. The ADVICE section finds trends in many of its data files, and uses them, along with averages, for analysis in greater depth. Messages to the user give trends and average values, as well as advice.

ADVICE section, using trends in voltage, recharge ratio, divergence and temperature--

AHO average (Ahoavg) is used for advice on workload. It is considered to be overly high above 14, high above 9, average around 7 and low below 3.

EOD pressure (Eodpress) is considered to be high above 100, and fairly high above 90. EOC pressure (Eocpress) is considered to be high above 110, and fairly high above 100.

Divergence (Div) is considered to be high above 0.8V (difference between high and low cells).

PREDICATES AVAILABLE IN ADVICE.PRG

advice(Bat,Choice)--with choice 1, advice/2 calls get_one/6 to get data, and recond/5 to analyze it. With choice 2, advice/2 calls get_two/7 to get data and chgchg/6 to analyze it. With choice 3, advice/2 calls get_three/4 to get data and chgwrk/3 to analyze it. advice/2 is called by more/2 (start.prg).

get_two(Trend2,Trend3,Div,Trend9,Eodpress,Eocpress,Bat)--The hand-crafted deviation factors are called (Dev2,Dev3,Dev5,Dev9). Then get_list/3 is called to read data from showf(2,_) (in-charge cell high voltage), showf(3,_) (recharge ratio), showf(5,_) (in-charge divergence), showf(9,_) (temperature), and showf(10,_) (cell pressure). Trend analysis is done on voltage, divergence and temperature by calling trend_analysis. find_div/5 is called to find average divergence. find_avg is called to find average cell pressure EOC (Eocpress) and EOD (Eodpress). get_two is called by advice/2.

get_three(Trend1,Trend4,Ahoavg,Bat)--Hand-crafted deviation factors are called (Dev1,Dev4). Then get_list/3 is called to read data from showf(1,_) (EOD battery voltage), showf(4,_) (EOD divergence) and Statf1,_) (AHO).
trend_analysis/3 is called to derive Trend1 of voltage. disp_trend/3 is called to find divergence trend. find_avg/2 is called to find average AHO. get_three is called by advice/2.

get_one(Trend1,Trend2,Trend3,Trend4,Trend5,Bat)--Hand-crafted deviation factors are called (Dev1,Dev2,Dev3,Dev4,Dev5). Then get_list/3 is called to read data from showf(1,_) (EOD battery voltage), showf(2,dat (in-charge cell high voltage), showf(3,_) (recharge ratio), showf(4,_) (EOD divergence) and showf(5,_) (in-charge divergence). trend_analysis/3 is called for voltages and recharge ratio. disp_trend/3 is called to find trends for divergence data. get_one is called from advice/2.

disp_trend(List,Trend,Dev)--break_list/4 is called to break the divergence List into highs (List1), lows (List3), and averages (List4). get_disp/3 is called to figure the divergence list (Divlist). Then trend_analysis/3 is called on Divlist. disp_trend/3 is called by get_three/4 and get_one/6.

get_disp(List1,List2,List3)--is a recursive function to find the differences between corresponding values of two lists, and put the differences in a third list. It is used to find the divergence between high-voltage cells and low-voltage cells in the data from 12 orbits. Missing data (-9999) in either of the comparands means that -9999 is written to the list of divergence. Comparison is done of the heads of the two lists, and the result is put in the head of the third list. Then get_disp/3 is called recursively on the tails of the lists. Halting condition for the recursion is the empty list. get_disp is called by disp_trend/3.

trend_analysis(List,Trend,Dev)--analyzes trends in data using two simple mathematical functions. find_w1/2 is called, which weights recent values more heavily, and sums the weighted values. Then find_w2/2 is called, which weights all the values equally, with the weight ((n+1)/2) chosen so that for a list of constant values, find_w1 and find_w2 give the same sum. Then eval_trend/3 is called using the difference of the two weight functions (Diff), and the hand-crafted deviation factor (see dev/6 below) appropriate to the set of data being analyzed, and returns the Trend. trend_analysis/3 is called by get_two/7, get_three/6, and get_one/4).

find_w1(Weight1,List)--given a list of n values, find_w1/2 calls multwt/2 to multiply the first by 1, the second value by 2, the third by 3, and so on till the nth value is multiplied by n. find_w1/2 uses counter 12 to keep track of the number of items in the list. Weightt1 is the sum of the individual products. find_w1 is called by trend_analysis/3.

multwt(Wt,[H|T])--checks for missing data (-99), then calls itself recursively on the tail of the list (T) until the empty list is reached. Each recursive call that does not find a missing data item increments the counter. Then as multwt/2 climbs out of the recursion, the value of the counter is multiplied by the head of the list (H) at that time, and is added to the weight, which has

been is initialized to 0 for the empty list. multwt/2 is called by find_w1/2.

find_w2(Weight2,List)--initializes a counter, then calls find_sum on List to find the sum of the values. Then a weight factor is figured which will make Weight2 (from find_w2) equal to Weight1 (from find_w1) if the List contains all one constant value. This weight factor is figured by (n+1)/2 for a list with n elements, and is multiplied by the sum. find_w2 is called by trend_analysis/3.

eval_trend(Dev,Diff,Trend)--takes the deviation factor, and the difference (Diff) between Weight1 and Weight2, and determines a trend. If the absolute value of Diff is less than Dev/3, the trend is none. For positive Diff, Diff greater than Dev/3 and less than Dev is considered to indicate the trend is slightly_up. Diff greater than Dev but less than 2 times Dev indicates the trend is up, and Diff greater than 2 times Dev indicates the trend is strongly_up. Similarly, negative Diff is slightly_down for Diff less than -Dev/3, down for Diff less than -Dev, and strongly_down for Diff less than 2 times Dev. Prolog syntax is used to advantage in this code, with atoms giving the trends. eval_trend/3 is called by trend_analysis/3.

recond(T1,T2,T3,T4,T5)--writes the trends (EOD voltage, in-charge high voltage, recharge ratio, EOD divergence and in-charge divergence) on the screen, and calls tell_1/5 to analyze the trends to determine whether reconditioning is advisable. recond/5 is called by advice/2.

chgchg(T2,T3,Div,T9,Eodpress,Eocpress)--writes the trends (in-charge voltage, recharge ratio, and temperature) on the screen, along with the averages (in-charge divergence, EOD pressure and EOC pressure). Then tell_2/6 is called to advise as to whether the charging scheme should be changed. chgchg/6 is called by advice/2.

chgwrk(T1,T4,Ahoavg)--writes EOD voltage and EOD divergence trends to the screen, along with the AHO average. Then tell_3/3 is called to advise if the workload should be changed. chgwrk/3 is called by advice/2.

tell_3(T1,T4,Ahoavg)--Six conditions are used by tell_3 to advise as to workload change.

1. EOD voltage down, EOD divergence down, AHO avg. less than 5 --loads probably too light.
2. EOD voltage strongly down, EOD divergence strongly down, AHO average less than 5 --losing capacity due to memory effect, try heavier load.
3. AHO average greater than 14 --possibly destructive overwork.
4. AHO average greater than 9 --heavy workload.
5. EOD voltage down or strongly down, EOD divergence down or strongly down, AHO average above 6 --losing capacity. Consider reconditioning.

6. none of the above --see no need to change workload.

tell_2(T2,T3,Div,T9,Eodpress,Eocpress)--Nine conditions are used by tell_2/6 to determine if the battery should have its charging scheme changed.

1. Temperature strongly up, EOD pressure greater than 100, EOC pressure greater than 110 --overcharging causing high pressure.
2. In-charge voltage strongly up --check for overcharging.
3. EOD pressure greater than 80, EOC pressure greater than 100 --charge rate may be too high.
4. EOD voltage strongly down, in-charge voltage strongly down --undercharging may be seriously affecting capacity.
5. EOD voltage down, in-charge voltage down --appears to be undercharged.
6. EOD voltage strongly down or down, in-charge voltage strongly down or down --undercharging may be causing loss of voltage.
7. EOD voltage down, in-charge divergence greater than 0.8 --undercharging may be causing loss of voltage.
8. EOD voltage strongly down, in-charge divergence greater than 8 --undercharging may be causing serious loss of voltage.
9. none of the above --see no need to change charge scheme.

tell_1(T1,T2,T3,T4,T5)--uses 11 conditions to determine if a battery needs to be reconditioned.

1. EOD voltage strongly down, in-charge divergence strongly up --reconditioning advised.
2. EOD voltage strongly down, in-charge voltage strongly down --recondition soon.
3. EOD divergence strongly up, in-charge divergence strongly up --strongly rising divergence indicates reconditioning.
4. Voltage trends strongly down or down, recharge ratio and divergence trends strongly up or up --reconditioning is indicated by all five trends.
5. Recharge ratio strongly up --reconditioning indicated.
6. In-charge voltage is down or strongly down, in-charge divergence is up or strongly up --reconditioning recommended.
7. EOD voltage down or strongly down, in-charge divergence up or strongly up --reconditioning indicated.
8. Both divergences up --consider reconditioning.
9. Recharge ratio up --consider reconditioning.
10. Both voltages down --consider reconditioning.
11. none of the above --see no need to recondition at present.

dev(Dev1,Dev2,Dev3,Dev4,Dev5,Dev9)--Data structure, listed as a fact, which contains the hand-crafted deviation factors. Each one is associated with a data file.

Dev1=2 - showf(1,_) - EOD battery voltage
Dev2=2 - showf(2,_) - in-charge high battery voltage
Dev3=0.02 - showf(3,_) - recharge ratio
Dev4=0.15 - showf(4,_) - EOD cell voltage divergence
Dev5=0.09 - showf(5,_) - in-charge cell divergence

Dev9=13 - showf(9,_) - battery temperature
These values are used in the weighting scheme to analyze trends.
To INCREASE sensitivity to trends, lower the values. These need not be integers, but must be greater than 0.
To change these values, put advice.prg in your favorite editor and revise the data structure dev/5. No other changes are necessary.

```
%  advice.prg derives trends to give further analysis on whether a
%  battery should be reconditioned, charging sheme changed or
%  workload changed.  See advice.doc for documentation.


advice(Bat,1) :-
      write($** ADVICE ON RECONDITIONING BATTERY: **$),nl,
      get_one(Trend1,Trend2,Trend3,Trend4,Trend5,Bat),
      recond(Trend1,Trend2,Trend3,Trend4,Trend5).

advice(Bat,2) :-
      write($** ADVICE ON CHANGING CHARGING REGIME OF BATTERY: ** $),nl,
      get_two(Trend2,Trend3,Div,Trend9,Eodpress,Eocpress,Bat),
      chgchg(Trend2,Trend3,Div,Trend9,Eodpress,Eocpress).

advice(Bat,3) :-
      write($** ADVICE ON CHANGING WORKLOAD OF BATTERY: **$),nl,
      get_three(Trend1,Trend4,Ahoavg,Bat),
      chgwrk(Trend1,Trend4,Ahoavg).

get_two(Trend2,Trend3,Div,Trend9,Eodpress,Eocpress,Bat) :-
      call(dev(_,Dev2,Dev3,_,_,Dev9)),
      get_list(Bat,2,List2),
      trend_analysis(List2,Trend2,Dev2),
      get_list(Bat,3,List3),
      trend_analysis(List3,Trend3,Dev3),
      get_list(Bat,5,List5),
      find_div(List5,_,Div,_,_),
      get_list(Bat,9,List9),
      trend_analysis(List9,Trend9,Dev9),
      get_list(Bat,10,List10),
      break_list(List10,Dlist,Clist,23),
      find_avg(Dlist,Eodpress),
      find_avg(Clist,Eocpress).

get_three(Trend1,Trend4,Ahoavg,Bat) :-
      call(dev(Dev1,_,_,Dev4,_,_)),
      get_list(Bat,1,List1),
      trend_analysis(List1,Trend1,Dev1),
      get_list(Bat,4,List4),
      disp_trend(List4,Trend4,Dev4),
      get_list(Bat,13,Listx),
      find_avg(Listx,Ahoavg).
```

```
get_one(Trend1,Trend2,Trend3,Trend4,Trend5,Bat) :-
      call(dev(Dev1,Dev2,Dev3,Dev4,Dev5,_)),
      get_list(Bat,1,List1),
      trend_analysis(List1,Trend1,Dev1),
      get_list(Bat,2,List2),
      trend_analysis(List2,Trend2,Dev2),
      get_list(Bat,3,List3),
      trend_analysis(List3,Trend3,Dev3),
      get_list(Bat,4,List4),
      disp_trend(List4,Trend4,Dev4),
      get_list(Bat,5,List5),
      disp_trend(List5,Trend5,Dev5).

disp_trend(List,Trend,Dev) :-
      break_list(List,List1,List2,12),
      break_list(List2,List3,List4,12),
      get_disp(List1,List3,Divlist),
      trend_analysis(Divlist,Trend,Dev).

get_disp([],[],[]).

get_disp([],List,[]).

get_disp(List,[],[]).

get_disp([H1|T1],[H2|T2],[Hr|T4]) :-
      (H1 == -9999; H2 == -9999),
      Hr = -9999,
      get_disp(T1,T2,T4).

get_disp([H1|T1],[H2|T2],[Hr|Tr]) :-
      Hr is H1 - H2,
      get_disp(T1,T2,Tr).

trend_analysis(List,Trend,Dev) :-
      find_w1(Weight1,List),
      find_w2(Weight2,List),
      Diff is Weight1 - Weight2,
      eval_trend(Dev,Diff,Trend).

find_w1(Weight1,List) :-
      ctr_set(12,1),
      multwt(Weight1,List).
```

```
multwt(Wt,[]) :-
     Wt is 0.

multwt(Wt,[H|T]) :-
     ifthen(H =\= -9999,ctr_inc(12,A)),
     multwt(Temp,T),
     B is A * H,
     ifthenelse(H == -9999,Wt is Temp,Wt is B + Temp).

find_w2(Weight2,List) :-
     ctr_set(10,0),
     find_sum(List,Sum),
     ctr_is(10,A),
     B is A + 1,
     C is B/2.0,
     Weight2 is C * Sum.

eval_trend(Dev,Diff,Trend) :-
     A is abs(Diff),
     A < Dev,
     Trend = none.

eval_trend(Dev,Diff,Trend) :-
     Diff > 0,
     Diff < (2 * Dev),
     Trend = slightly_up.

eval_trend(Dev,Diff,Trend) :-
     Diff > 0,
     Diff =< (3 * Dev),
     Trend = up.

eval_trend(Dev,Diff,Trend) :-
     Diff > 0,
     Diff > (3 * Dev),
     Trend = strongly_up.

eval_trend(Dev,Diff,Trend) :-
     Diff < 0,
     A is -2 * Dev,
     Diff > A,
     Trend = slightly_down.

eval_trend(Dev,Diff,Trend) :-
     Diff < 0,
     A is -3 * Dev,
     Diff > A,
     Trend = down.
```

```
eval_trend(Dev,Diff,Trend) :-
      Diff < 0,
      A is -3 * Dev,
      Diff =< A,
      Trend = strongly_down.

record(T1,T2,T3,T4,T5) :-
      write($EOD voltage trend is $),write(T1),nl,
      write($In-charge voltage trend is $),write(T2),nl,
      write($Recharge ratio trend is $),write(T3),nl,
      write($EOD divergence trend is $),write(T4),nl,
      write($In-charge divergence trend is $),write(T5),nl,
      tell_1(T1,T2,T3,T4,T5).

chgchg(T2,T3,Div,T9,Eodpress,Eocpress) :-
      write($In-charge voltage trend is $),write(T2),nl,
      write($Recharge ratio trend is $),write(T3),nl,
      write($In-charge divergence avg. over last 12 orbits is $),
      write(Div),nl,
      write($Avg. of cell pressures at EOD over last 12 orbits is $),
      write(Eodpress),nl,
      write($Avg. of cell pressures at EOC over last 12 orbits is $),
      write(Eocpress),nl,
      write($Temperature trend is $),write(T9),nl,
      tell_2(T2,T3,Div,T9,Eodpress,Eocpress).

chgwrk(T1,T4,Ahoavg) :-
      write($EOD voltage trend is $),write(T1),nl,
      write($EOD divergence trend is $),write(T4),nl,
      write($Average amp-hours-out over last 12 orbits is $),
      write(Ahoavg),nl,
      tell_3(T1,T4,Ahoavg).

tell_3(down,down,Ahoavg) :-
      Ahoavg < 5,
      write($**Battery loads are probably too light--memory effect$),nl,
      write($  is indicated.$).

tell_3(strongly_down,strongly_down,Ahoavg) :-
      Ahoavg < 5,
      write($**Battery is losing capacity due to memory effect. Try a$),
      nl,write($  deeper DoD first$),nl.

tell_3(_,_,Ahoavg) :-
      Ahoavg > 14,
      write($**Battery is overworked--may result in destructive damage.$),
      nl.
```

```
tell_3(_,_,Ahoavg) :-
     Ahoavg > 9,
     write($**Battery workload seems heavy at $),
     write(Ahoavg),write($AH0.$),
     nl.

tell_3(T1,T4,Ahoavg) :-
     (T1 = down; T1 = strongly_down),
     (T4 = down; T4 = strongly_down),
     Ahoavg > 6,
     write($**Battery is losing capacity.  Consider reconditioning.$),nl.

tell_3(_,_,_) :-
     write($See no need to change workload.$),nl.

tell_2(_,_,_,strongly_up,Eodpress,Eocpress) :-
     Eodpress > 100,
     Eocpress > 110,
     write($**Overcharging caused high pressure. Check charge limit$),nl,
     write($  circuitry, or change charge regime immediately.$),nl.

tell_2(Trend2,_,_,_,_,_) :-
     (Trend2 is strongly_up),
     write($**Check for overcharging; voltage is high.$).

tell_2(_,_,_,_,Eodpress,Eocpress) :-
     Eodpress > 80,
     Eocpress > 100,
     write($**Charge rate may be too high.  Pressure high at EOC, though$),
     nl,write($  gas recombination lowers pressure during discharge.$),nl.

tell_2(strongly_down,strongly_down,_,_,_,_) :-
     write($**Battery undercharging may be seriously affecting battery$),
     nl,write($  capacity for work.$),nl.

tell_2(down,down,_,_,_,_) :-
     write($**Battery appears to be undercharged.$),nl.

tell_2(T2,T3,_,_,_,_) :-
     (T2 is strongly_down; T2 is down),
     (T3 is strongly_down; T3 is down),
     write($**Undercharging may be causing loss of voltage.$),nl.

tell_2(down,_,Div,_,_,_) :-
     Div < 0.8,
     write($**Undercharging may be causing loss of voltage.$),nl.

tell_2(strongly_down,_,Div,_,_,_) :-
     Div < 0.8,
     write($**Undercharging may be causing serious loss of voltage.$),nl.
```

```
tell_2(_,_,_,_,_,_) :-
     write($See no need to change charging scheme.$),nl.

tell_1(strongly_down,_,_,_,strongly_up) :-
     write($**Reconditioning advised to correct failing capacity.$),nl.

tell_1(strongly_down,strongly_down,_,_,_) :-
     write($**Recondition soon before battery fails.$),nl.

tell_1(_,_,_,strongly_up,strongly_up) :-
     write($**Strongly rising divergence suggests reconditioning soon.$),
     nl.

tell_1(T1,T2,T3,T4,T5) :-
     (T1 is strongly_down; T1 is down),
     (T2 is strongly_down; T2 is down),
     (T3 is strongly_up; T3 is up),
     (T4 is strongly_up; T4 is up),
     (T5 is strongly_up; T5 is up),
     write($**Reconditioning is indicated by all five trends.$),nl.

tell_1(_,_,strongly_up,_,_) :-
     write($**Recharge ratio indicates serious loss of efficiency--$),
     nl,write($  suggest reconditioning soon.$),nl.

tell_1(_,T2,_,_,T5) :-
     (T2 is down; T2 is strongly_down),
     (T5 is up; T5 is strongly_up),
     write($**Reconditioning recommended.  Too many cells are migrating$),
     nl,write($  to low values.$),nl.

tell_1(T1,_,_,_,T5) :-
     (T1 is down; T1 is strongly_down),
     (T5 is up; T5 is strongly_up),
     write($**Consider reconditioning to correct poor performance.$),nl.

tell_1(_,_,_,up,up) :-
     write($**Consider reconditioning to correct divergence.$),nl.

tell_1(_,_,up,_,_) :-
     write($**Consider reconditioning to correct rising recharge ratio.$),
     nl.

tell_1(down,down,_,_,_) :-
     write($**Battery is losing capacity; consider reconditioning.$),nl.

tell_1(_,_,_,_,_) :-
     write($See no need to recondition at present$),nl.

dev(2.0,2.0,0.02,0.05,0.09,4.3).
```

Showpak.prg contains the DECISION SUPPORT system, which gives the 12 plots which are available to the user upon selecting Plots and Graphs from the Main Menu. The graphical primitives from Grafpak.prg are used in the plots. Showpak contains a few supporting data handling primitives and the titles, horizontal and vertical captions used in the plots.

12 Plots in the DECISION SUPPORT package:

1) EOD battery voltage	low plotted	26.4V.
	high plotted	28.6V
	scale	.2V
2) In-charge high cell voltage	low plotted	32.0V
	high plotted	34.2V
	scale	.2V
3) recharge ratio	low plotted	1.010
	high plotted	1.055
	scale	.005
4) EOD divergence	low plotted	1.05V
	high plotted	1.55V
	scale	.05V (cell)
5) In-charge divergence	low plotted	1.35V.
	high plotted	1.55V.
	scale	.02V. (cell)
6) EOD cell V. divergence	low plotted	1.05V.
	high plotted	1.55V.
	scale	.05V. (cell)
7) In-charge cell divergence	low plotted	1.36V.
	high plotted	1.56V.
	scale	.02V. (cell)
8) Temperature this orbit	low plotted	-4C
	high plotted	11C
	scale	1C
9) Temp. avg.	low plotted	-4C
	high plotted	11C
	scale	1C
10) Cell Pressure	low plotted	30

	EOD and EOC	high plotted	140
		scale	10 units
11)	Time on trickle charge	low plotted	8
		high plotted	28 minutes
		scale	2 minutes
12)	Reconditioning current	low plotted	0 A.
		high plotted	14 A.
		scale	1 A.

The above numbers can easily be manipulated to change range or scale of the plots, but must remain integers. These changes can be made in Showpak.prg by editing the correct 'show' data structure. 'show' has the following format:

1. N - the number of the plot (1 to 12)
2. Vertcap - vertical caption
3. Horizcap - horizontal caption
4. Title - graph title
5. H - heigth of graph
6. W - width of scale
7. Base - lower bound for data points plotted
8. Top - upper bound for data points plotted
9. Hscale - horizontal scale of values
10. Vscale - vertical scale of values
11. Start - start point of graph

Changing the number of orbits plotted from 12 is more difficult, and will require chasing down many 11's and 12's, as well as rewriting the file handler portion of the code.

Colors are
- 1 = blue
- 2 = green
- 3 = light blue
- 4 = red
- 5 = pink
- 6 = gold
- 7 = white
- 8 = gray

Colors are set in predicates "wa(n,color)", where n is the number of characters of that color, and color is one of the numbers to the left.

Symbols for the graphs are referrenced by their ASCII code.

THE PREDICATES AVAILABLE IN SHOWPAK.PRG

show_view(N,Bat,Orbit)--N specifies which plot is done, and Bat specifies which battery is displayed. Orbit gives the current orbit number. The data structure 'show', discussed above, is called to get all the captions and values needed for plot N. show_view/3 calls get_list/3 to read the appropriate data file.

Then graphplus/6 and plot/8 (in Grafpak.prg) are called to plot the display. Besides the information in 'show', only the color and symbol need to be passed. The present orbit is then written to the plot, if applicable. show_view/3 is called by execute/2, the Graphics Menu.

show_view/3 -- There are four cases; plot 10, plot 12, plot 4 and 5, and the rest. Plot 10 displays 2 lines, plot 12 has the orbit no. preceeding its data, plot 4 and 5 display 3 lines each. For plots 4,5 and 10, breaklist/4 is called to break the data into lists, each one to be plotted on the same graph. plot/8 is then called for each list with a different color and symbol.

write_orbit/0 -- writes 'Orbit' on the horizontal axis. Called by show_view/3.

write_orbit/1 -- writes 'Orbit' and the last orbit on the horizontal axis. Called by show_view/3.

show/11 -- a data structure containing the captions and values needed to display the graphs. Itemized above.

```
%  Showpak--contains the decision support system, i.e., show_view/3 to
%  build the 12 plots which support analysis in Status and Advice.
%  See showpak.doc fro further documentation.

show_view(10,Bat,Orbit) :-
      get_list(Bat,10,List),
      call(show(10,Vcap,Hcap,Title,H,W,Base,Top,Hscale,Vscale,Start)),
      graphplus(H,W,Vcap,Hcap,Title,Bat),
      break_list(List,List1,List2,23),
      plot(List1,Hscale,Vscale,Base,Top,3,Start,254),
      plot(List2,Hscale,Vscale,Base,Top,4,Start,4),
      write_orbit(Orbit),!.

show_view(12,Bat,Orbit) :-
      get_list(Bat,12,[R,List]),
      call(show(12,Vcap,Hcap,Title,H,W,Base,Top,Hscale,Vscale,Start)),
      graphplus(H,W,Vcap,Hcap,Title,Bat),
      plot(List,Hscale,Vscale,Base,Top,4,Start,254),
      write_orbit(R).

show_view(V,Bat,Orbit) :-
      (V == 4; V == 5),
      get_list(Bat,V,List),
      call(show(V,Vcap,Hcap,Title,H,W,Base,Top,Hscale,Vscale,Start)),
      graphplus(H,W,Vcap,Hcap,Title,Bat),
      break_list(List,List1,List2,12),
      break_list(List2,List3,List4,12),
      plot(List1,Hscale,Vscale,Base,Top,3,Start,254),
      plot(List3,Hscale,Vscale,Base,Top,4,Start,4),
      plot(List4,Hscale,Vscale,Base,Top,7,Start,42),
      write_orbit,!.

show_view(V,Bat,Orbit) :-
      get_list(Bat,V,List),
      call(show(V,Vcap,Hcap,Title,H,W,Base,Top,Hscale,Vscale,Start)),
      graphplus(H,W,Vcap,Hcap,Title,Bat),
      plot(List,Hscale,Vscale,Base,Top,4,Start,254),
      (((V == 8;V == 6;V == 7),write_orbit(Orbit));
      write_orbit),!.

write_orbit :-
      tmove(24,3),
      wa(5,2),
      write($Orbit$).
```

```
write_orbit(Orbit) :-
      tmove(24,0),
      wa(10,7),
      write($Orbit $),
      write(Orbit).

show(1,[28.6,28.4,28.2,28.0,27.8,27.6,27.4,27.2,27.0,26.8,26.6,26.4],
Hcap1,'   EOD VOLTAGE FOR LAST 12 ORBITS',13,65,26.4,28.6,5,0.2,11).

show(2,[34.2,34.0,33.8,33.6,33.4,33.2,33.0,32.8,32.6,32.4,32.2,32.0],
Hcap1,'   HIGH VOLTAGE DURING CHARGE FOR LAST 12 ORBITS',
13,65,32.0,34.2,5,0.2,11).

show(3,[1.055,1.050,1.045,1.040,1.035,1.030,1.025,1.020,1.015,1.010],
Hcap1,'   RECHARGE RATIO FOR LAST 12 ORBITS',11,65,1.010,1.055,5,0.005,11).

show(4,[1.55,1.50,1.45,1.40,1.35,1.30,1.25,1.20,1.15,1.10,1.05],Hcap1,
'   EOD HIGH(blue) - LOW(red) - AVG(wh) FOR LAST 12 ORBITS',
12,65,1.05,1.55,5,0.05,11).

show(5,[1.55,1.53,1.51,1.49,1.47,1.45,1.43,1.41,1.39,1.37,1.35],Hcap1,
'   HIGH(blue) - LOW(red) - AVG(wh) - DURING CHARGE  LAST 12 ORBITS',
12,65,1.35,1.55,5,0.02,11).

show(6,[1.55,1.50,1.45,1.40,1.35,1.30,1.25,1.20,1.15,1.10,1.05],
[1,' ',6,' ',11,' ',16,' ',21,' ',26,' '],
'   23 EOD CELL VOLTAGES FOR LAST ORBIT',13,62,1.05,1.55,2,0.05,11).

show(7,[1.56,1.54,1.52,1.50,1.48,1.46,1.44,1.42,1.40,1.38,1.36],
[1,' ',6,' ',11,' ',16,' ',21,' ',26,' '],
'   23 HIGH CHARGE CELL FOR LAST ORBIT',13,62,1.36,1.56,2,0.02,11).

show(8,['11C','10C','9C','8C','7C','6C','5C','4C','3C','2C','1C','0C','-1C',
'-2C','-3C','-4C'],[0,10,20,30,40,50,60,70,80,90,100],
'   TEMPERATURE IN LAST ORBIT, EACH 2 MIN ',17,60,-4,11,1,1,12).

show(9,['11C','10C','9C','8C','7C','6C','5C','4C','3C','2C','1C','0C','-1C',
'-2C','-3C','-4C'],Hcap1,
'   AVG. TEMPERATURE IN EACH ORBIT, LAST 12 ORBITS',17,65,-4,11,5,1,11).

show(10,[140,130,120,110,100,90,80,70,60,50,40,30],
[1,' ',6,' ',11,' ',16,' ',21,' ',26,' '],
'   CELL PRESSURE EOD (red), EOC (blue), DURING LAST ORBIT',
13,62,30,140,2,10,11).

show(11,[28,26,24,22,20,18,16,14,12,10,8],Hcap1,
'   TIME ON TRICKLE CHARGE, LAST 12 ORBITS',12,65,8,28,5,2,11).
```

```
show(12,[14,13,12,11,10,9,8,7,6,5,4,3,2,1,0],
[0,10,20,30,40,50,60,70,80,90,100],
'   CURRENT DURING CAP. TEST/RECOND, EACH 2-MIN, 0 => NO RECOND',
16,60,0,14,1,1,12).
```

Grafpak.prg contains graphic primitives and supporting predicates for Showpak.prg to use in giving the user plots and charts to aid in decision support. The procedures are written to be general-purpose so that plots may be altered easily. New plots may be developed by the programmer using these primitive operations.

Arity/Prolog graphical primitives

Parameters in plot are

(List,	list of values to be plotted
Hscale,	distance horizontally between plotted points
Vscale,	vertical scale of values
Base,	lowest value plotted
Top,	highest value plotted
Color,	color of the plotted points
Coord,	the horizontal positioning factor, generally starts with 11 (columns from left-hand side of screen), increments by "num" with each point (Start)
Obj)	ASCII code for symbol to be plotted

Parameters in graphplus are

(H,	Rows in height (1 more than number of values wanted)
W,	Columns in width (5 more than the spaces needed to plot values--see "coord"
Vcap,	Vertical caption
Hcap,	Horizontal caption
Title)	Title written above plot

Built-in Graphics Primitives:

tmove(x,y)-- moves the cursor to row x (rows are numbered 0 to 24 beginning at the top) and column y (columns are numbered 0 to 79 beginning at left).

wc(n,char)-- prints n copies of char (given as an ASCII code or within single quotes) starting at the cursor.

wa(n,color)--prints the next n characters in color (given as a number).

1 = blue	2 = green	3 = lt. blue	4 = red
5 = pink	6 = gold	7 = white	8 = gray

Characters can also be made flashing or inverse, in any of the colors, using this command. To see the attributes that are available, with the Arity interpreter running, type "['table.ari']." and when Arity returns "yes" type "table."

THE PREDICATES IN GRAFPAK.PRG

graphplus(H,W,Vcap,Hcap,Title,Bat)--creates the basic graph with height H and width W, vertical caption Vcap (usually the scale), horizontal caption Hcap, and title Title including the battery number, Bat. H should be chosen to be 1 greater than the number of rows (which is the number of possible values that will be plotted). W should be chosen to be 5 wider than the spaces needed to plot the values. See plot/8 for more information.

graphplus/6 calls graph(H,W,Hcap) to build the skeleton along with the horizontal caption, then calls write_vert(Vcap), and write_title(Title,R,Bat) to fill in the captions. write_header is also called to write information about missing data and data out of range at the the top of the graph.

graphplus/6 is called by show_view/3 in Showpak.prg, with the proper parameters of size and captions.

graph(H,W,Hcap)--builds the skeleton graph of height H and width W, using vertical/2 for the vertical graphics characters making the left-hand side of the graph, and horizontal/2 for the lower edge. graph/2 is called by graphplus/6.

vertical(A,W2)--uses graphics characters 180 (ticks) for the vertical axis and 250 for the dots across the inside of the graph.

horizontal(B,List)--uses graphics characters 196 and 194 for the bottom edge of the graph. The tick marks are placed every five spaces. The elements of the List, containing the horizontal captions, are alined with the ticks but one line under.

write_vert(L,P)--writes the list L along the left-hand side of the graph. Calls write_scale(L) which writes the list recursively.

write_scale(L)--writes the vertical caption recursively from the list L. The empty list ([]) halts the recursion.

write_title(S,P,Bat)--writes the Title along with the Bat, battery number, above the graph. S is the Title and P is the row position.

plot(List,Hscale,Vscale,Base,Top,Color,Coord,Obj)--Plots the values in list List on the basic graph produced by graphplus/6. Hscale is the horizontal distance (in columns of the screen) between the points plotted. Vscale is the vertical scale of the graph. Base is the lowest value plotted, and Top is the highest value plotted. An attempt to plot any value outside this range will result in out of range values being plotted at the top of the graph and the actual values being written, in order at the top of the

screen. Top minus Base divided by Vscale will equal the number of rows upon which the graph is plotted. Color is the color of the plotted point. Coord is the horizontal positioning factor, as the plot moves across the screen from left to right. Coord ordinarily starts at column 11, and increments by either 1 or 5. Obj is the ASCII code for the symbol which will be plotted for each data point.

plot/8 calls plot_point/8 to recursively plot the points from the list L. When plot_point/8 is done, plot/8 changes the color back to white and moves the cursor to the top left-hand corner.

plot/8 is called by show_view/3 in Showpak.prg with the data in the List and specific parameters for the plotting.

plot_point([Head|Tail],Hscale,Vscale,Base,Top,Coord,Color,Obj)--plot_point/8 recursively plots the list of points, divided into Head and Tail, then calls itself on the Tail, having updated Coord for the next point. The empty list halts the recursion. Hscale, Vscale, Base, Top, Coord, Color and Obj are the same as in plot/8.

plot_point/8 calls place/7 to find the correct place for the plotted point, and plot it there in the correct color.

place(Head,Vscale,Base,Top,Coord,Color,Obj)--figures the correct place for the plotted point, moves the cursor there, and plots the point. The value for missing data causes an asterisk in gold to be plotted on the low boundary line of the graph. place/7 truncates values as it plots them. When values are out of range, place/7 plots a symbol at the top boundary of the graph and writes the value at the top of the screen.

place/7 is called by plot_point/8.

```
%  grafpak.prg--contains graphic primitives and supporting predicates.
%  graphplus builds a graph skeleton given height, width and captions.
%  plot draws the data points on the graph.
%  See Grafpak.doc for further documentation.


graphplus(H,W,Vcap,Hcap,Title,Bat) :-
      graph(H,W,Hcap),
      Q is 24 - H,
      write_vert(Vcap,Q),
      R is Q - 2,
      write_title(Title,R,Bat),
      write_header,
      tmove(1,1),
      R1 is R + 1,
      ctr_set(2,R1),
      ctr_set(10,0).

write_header :-
      tmove(2,0),
      wa(80,6),wc(1,236),
      tmove(2,2),write($= MISSING DATA,  $),
      wc(1,254),tget(_,X),X1 is X + 2,
      tmove(2,X1),
      write($= VALUE OUT OF RANGE, LISTED ON NEXT LINE$).

graph(H,W,Hcap) :-
      cls,
      H1 is 24 - H,
      W1 is W - 1,
      vertical(H1,W1),
      horizontal(W,Hcap),
      tmove(23,5),
      wa(1,2),
      wc(1,192).

vertical(A,W2) :-
      ctr_set(0,A),
      repeat,
      ctr_inc(0,A1),
      tmove(A1,5),
      wa(1,2),
      wc(1,180),
      tmove(A1,6),
      wa(W2,2),
      wc(W2,250),
      A1 == 22.
```

```
horizontal(B,List) :-
     B1 is (B - 2)//5,
     ctr_set(0,1),
     tmove(23,6),
     wa(1,2),
     wc(1,196),
     tmove(23,7),
     repeat,
     list_iterate(H,List),
     ctr_inc_1(0,C),
     wa(4,2),
     wc(4,196),
     D is 5 * (C + 1) + 1,
     tmove(23,D),
     wa(1,2),
     wc(1,194),
     tmove(24,D),
     wa(10,2),
     write(H),
     E is D + 1,
     tmove(23,E),
     C == B1,
     F is (B - 2) mod 5,
     wa(10,2),
     ifthen(F @> 0, wc(F,196)).

ctr_inc_1(Cnt,N) :- ctr_inc(Cnt,N),!.

list_iterate(H,[H|T]).
list_iterate(H,[_|T]) :-
     list_iterate(H,T).
list_iterate('',[]).

write_vert(L,P) :-
     tmove(P,0),
     write_scale(L).

write_scale([]).
write_scale([H|T]) :-
     wa(5,2),
     write(H), nl,
     write_scale(T).

write_title(S,P,Bat) :-
     tmove(P,6),
     wa(80,7),
     write($BATTERY $),write(Bat),
     write(S).
```

```
plot(List,Hscale,Vscale,Base,Top,Color,Coord,Obj) :-
      plot_point(List,Hscale,Vscale,Base,Top,Coord,Color,Obj),
      tmove(1,1),
      wa(1,7).

plot_point([],_,_,_,_,_,_,_).
plot_point([Head|Tail],Hscale,Vscale,Base,Top,Coord,Color,Obj) :-
      place(Head,Vscale,Base,Top,Coord,Color,Obj),
      Coord1 is (Coord + Hscale),
      plot_point(Tail,Hscale,Vscale,Base,Top,Coord1,Color,Obj).

place(-9999,_,_,_,Coord,_,_) :-
      tmove(23,Coord),
      wa(1,6),
      wc(1,236).

place(Head,Vscale,Base,Top,Coord,Color,Obj) :-
      (Head < Base; Head > Top),
      ctr_is(2,R1),
      tmove(R1,Coord),
      wa(1,6),
      wc(1,Obj),
      ctr_is(10,R2),
      tmove(3,R2),
      wa(5,6),
      write(Head),write($  $),
      tget(_,R3),ctr_set(10,R3).

place(Head,Vscale,Base,Top,Coord,Color,Obj) :-
      Q1 is (Head - Base)/Vscale,
      Q is round(Q1,0),
      Level is 22 - integer(Q),
      tmove(Level,Coord),
      wa(1,Color),
      wc(1,Obj).

%place(Head,Vscale,Base,Top,Coord,Color,Obj) :-
%     Q1 is (Head - Base)/Vscale,
%     Q is round(Q1,0),
%     Level is 22 - integer(Q),
%     tmove(Level,Coord),
%     wa(1,Color),
%     wc(1,Obj).
```

Utility.prg is a collection of general Prolog predicates which are used in one or more of the routines comprising NICBES.

Data Files in the Expert System:

All data files are loaded into PROLOG as facts in the following format:

```
showf(N,List).   N = 1 to 13 for showf(N).dat
curf(N,List).    N = 1 to 3 for curf(N).dat
fault([Flag,Type]).  for fault.dat
```

List for show files contains 6 sublists, one for each battery, seperated by commas. For example, in PROLOG, showf1.dat is loaded as:

```
showf(1,[[27.0,26.4,27.0,26.4,-9999,27.0,26.4,27.0,26.4,26.9,26.8,27.2],
     [26.8,27.0,26.8,27.0,26.8,26.6,27.0,26.8,27.0,27.0,26.4,27.0],
     [27.2,27.2,27.2,27.0,26.8,26.8,26.8,26.8,26.8,27.1,26.5,26.0],
     [27.5,26.9,26.9,26.9,26.9,26.8,26.8,27.0,26.8,27.0,27.0,26.7],
     [27.4,27.0,27.1,27.0,27.0,27.3,27.2,27.0,26.9,26.8,26.8,26.8],
     [27.0,26.4,27.0,26.4,26.4,27.0,26.4,27.0,27.0,281,27.2,27.5]]).
```

Actual data points are reals with 5 decimal places. -9999.0 represents missing data points. There are 12 columns, one for each orbit in chronological order from oldest to latest.

An example of the current data files is curf2.dat:

```
curf(2,[Phase,Day_min,[13 SPA Currents],[3 Bus Currents],
       [6 Average Battery Temperatures]]).
```

Built-in ARITY PROLOG Functions:

nl--gives a new line (like carriage return).

cls--clears the screen.

Many of the functions use the ARITY PROLOG counters. There are 32 counters (0 to 31) which are accessible from any procedure in any file. The operations on a counter are:

ctr_set(n,m)	set counter n to value m
ctr_is(n,A)	A is instantiated to the value of counter n
ctr_inc(n,A)	counter n is incremented, and A is instantiated to the previous value
ctr_dec(n,A)	counter n is decremented, and A is instantiated to the previous value

Underscores (_) appearing in place of arguments signifies an anonymous

variable--we do not care what the value is and will not be using it at this time. This saves the system work.

ifthen and ifthenelse use the following format:
ifthen(X,Y) => if X then Y.
ifthenelse(X,Y,Z) => if X then Y else Z.

var(X)--succeeds if X is a variable.

integer(X)--converts X to an integer.

round(X,N)--rounds X to N decimal places.

read_line(H,String)--reads string from file H. When H == 0, reads string from keyboard.

PREDICATES AVAILABLE IN UTILITY.PRG

get_list(Bat,N,List)--N tells which show file. get_list calls the particular 'showf' fact which was loaded in eval_flag(0) (start.prg) and which is described above. find_nth is called next to find the data list corresponding to Bat, the battery of interest. This list is returned to the variable List. get_list/3 is called from complete/1, status.prg, advice.prg and showpak.prg.

find_nth/4--given a list, find_nth/4 finds the Nth element of the list. It is called by get_list/3.

append(List1,List2,List3)--appends List2 to List1, with the result in List3. append/3 calls itself recursively, with the empty list the halting condition.

break_list(List,List1,List2,N)--breaks List into two lists. The first N items are put in List1, and the remainder into List2. After setting counter 1 with N, break_list/4 calls steal_item to recursively build the lists. break_list/4 is called by show_view/3 (grafpak.prg), disp_trend/3 (advice.prg) and find_div/5 (status.prg).

steal_item([H|T],List1,List2)--recursively traverses the tail of its first argument, decrementing the counter, until the counter is 0, or the first argument is the empty list. Then List2 is set to the remaining portion of the original list (or the empty list), and List1 is built as steal_item/3 climbs out of the recursion. steal_item/3 is called by break_list/4.

find_avg(List,Avg)--finds the average of a list of numbers. Counter 10 is set to count the number of items in the list. find_sum is called to sum them, then the average is found. find_avg is called by get_data/5 and find_div/5 (status.prg).

find_sum([H|T],Sum)--recursively finds the sum of a list of numbers. The head of the list (H) is checked for missing data (-9999), counter 10 is incremented, then find_sum/2 is called on the tail (T) of the list.

The empty list halts the recursion, then the sum is found as find_sum climbs out of the recursion. A missing data item is left out of the sum, and a message is printed. find_sum/2 is called from find_avg/2, find_w2/
(advice.prg) and loadbank_fail/0 (faultd.prg).

sequence(Orbit)--creates the horizontal axis for those graphs using the 12 latest orbits. It calls seq/3 to make this list and horiz_place/1 to pu
it in the data structure show/11. It is called by eval_flag(1) (start.prg).

seq(H,T,[H|Z])--Builds the list of last 12 orbits recursively from the latest orbit, T and the first orbit in this series, H. Called by sequence/1.

horiz_place(List)--searches the data structures show/11 to find ones with variable Hcap1, the horizontal axis. It retracts these and then asserts them to the data base with the List in the place of the variable. Called by sequence/1.

cut_retract/0--retract fact and cut. Called by horiz_place/1.

check(X,N)--used for checking the user input to menus. If the user selection X is >0 and <= N (upper bound), the screen is cleared and program continues. If not, a message is written to the user saying 'YOUR CHOICE IS OUT OF RANGE!'. The user can then input a proper selection. Called by Status, Advice and Graphics menus in start.prg.

check2(X,N)--used for checking the user input to Main Menu in start.prg. If the user selection X is >0 and either X <= N (upper bound) or equal to 4, the screen is cleared and program continues. If not, a message is written to the user saying 'YOUR CHOICE IS OUT OF RANGE!'. The user can then input a proper selection.

check1(X)--used for checking the user yes/no responses to menus. If the user input is either yes or no, the screen is cleared and program continues. If not, a message is written saying 'ANSWER IS OUT OF RANGE!'. The user can then re-enter a correct reply. Called by eval_flag(1) (start.pr
complete(M)--calls get_list/3 to look at show file 1. missing/2 returns the number of missing data points in show file 1. If there are more than 4 missing values, M is set to 1 so only Graphics may be called from the Main Menu. Else M is set to 4 and all options from the Main Menu will be valid. Called from eval_flag(0) (start.prg).

missing(List,N)--returns the number of missing data points,N from the List. Called by complete/1.

reader(Atom)--reads input from the keyboard and returns it as an atom. Called by all menus in start.prg.

one_assert(X)--if X is already in the data base, no action. Else the fact X is asserted to the data base.

```
%  Utility  contains general functions which are used in one or more
%  of the routines comprising NICBES.

get_list(Bat,N,List):-
     call(showf(N,L)),
     find_nth(L,1,Bat,List).

find_nth([H|T],N,N,H).

find_nth([H|T],N1,N,X) :-
     M is N1 + 1,
     find_nth(T,M,N,X).

break_list(List,List1,List2,N) :-
     ctr_set(1,N),
     steal_item(List,List1,List2).

steal_item([],List1,List2) :-
     List1 = [],
     List2 = [].

steal_item(List,List1,List2) :-
     ctr_is(1,M),
     M == 0,
     List1 = [],
     List2 = List.

steal_item([H|T],List1,List2) :-
     ctr_dec(1,_),
     steal_item(T,Temp,List2),
     List1 = [H|Temp].

find_avg(List,Avg) :-
     ctr_set(10,0),
     find_sum(List,Sum),
     ctr_is(10,Num),
     Avg1 is Sum/Num,
     Avg is round(Avg1,4).

find_sum([],Sum) :-
     Sum is 0.0.

find_sum([H|T],Sum) :-
     ctr_inc(10,_),
     find_sum(T,Temp),
     ifthenelse(H == -9999,(Sum is Temp,ctr_dec(10,_)),Sum is H + Temp).
```

```
sequence(Orbit) :-
     H is Orbit - 11,
     Orb is Orbit + 1,
     seq(H,Orb,List),
     horiz_place(List).

seq(T,T,[]).

seq(H,T,[H|Z]) :-
     H1 is H + 1,
     seq(H1,T,Z).

horiz_place(List) :-
     call(show(N,Vcap,Hcap,Title,H,W,Base,Top,Hscale,Vscale,Start)),
     var(Hcap),
     cut_retract(show(N,Vcap,Hcap,Title,H,W,Base,Top,Hscale,Vscale,Start)),
     asserta(show(N,Vcap,List,Title,H,W,Base,Top,Hscale,Vscale,Start)),
     fail.

horiz_place(_) :- !.

cut_retract(X) :- retract(X),!.

check(X,N) :-
     X > 0,
     X =< N,
     cls,!.

check(X,N) :-
     cls, nl,nl,
     write($        YOUR CHOICE IS OUT OF RANGE! $),nl,nl,
     !,fail.

check2(X,N) :-
     X > 0,
     (X =< N; X == 4),
     cls,!.

check2(X,N) :-
     cls, nl,nl,
     write($        YOUR CHOICE IS OUT OF RANGE! $),nl,nl,
     !,fail.

check1(X) :-
     (X == 'yes'; X == 'no'),!.

check1(X) :-
     cls,nl,nl,
     write($  ANSWER IS OUT OF RANGE! $),
     nl,nl,!,fail.
```

```
complete(M) :-
     get_list(1,1,List),
     ctr_set(1,0),
     missing(List,N),
     ifthenelse(N < 5, M is 4,
          (write($ Not enough orbital data for NICBES analysis.$),
          nl,write($ Check data files.  You may run Graphics.$),
          M is 1)).

missing([],N) :- ctr_is(1,N).

missing([H|T],N) :-
     ifthen(H == -9999,ctr_inc(1,_)),
     missing(T,N).

reader(Atom) :-
     read_line(0,String),
     (int_text(Atom,String);
      atom_string(Atom,String)),!.

one_assert(X) :-
     X.

one_assert(X) :-
     asserta(X).
```

APPENDIX C

TEST PROCEDURES

1.0 Power on the IBM-PC AT and Printer.

2.0 Run the Data-Handler.
- A. Data-Handler is in directory USR.
- B. Enter 'data_hdl' to execute Data-Handler.
- C. Check telemetry flow across RS232.
- D. Messages printed to screen:
  "Starting first full orbit",
  "#. ORBIT = N" - every 96 minutes.
- E. If a fault is detected, message printed to screen:
  "Fault detected; exiting!".
- F. If no communications being received, message printed to screen; "No communication for 3 minutes; exiting!".
- G. If 5 consecutive incomplete telemetry runs, message printed to screen; "Received 5 consecutive incomplete telemetry runs; exiting!".
- G. When ready to quit Data-Handler enter '^C'. Message printed: "Interrupt caught; exiting!".

3. Verify and Archive Data-Handler Output.
- A. View showf#.dat (1 to 13),curf#.dat (1 to 3) and fault.dat for correct format.
- B. Archive data files under directory DATAFILES, subdirectory 'DATE' where DATE is current date.

4. Run Expert System.
- A. Expert System is in directory NICBES.
- B. Enter 'api' to execute Expert System.
- C. View Fault Diagnosis.
  - a. Print Fault Diagnosis Report.
- D. View Main Menu and make all selections.
- E. Select battery.
- F. View Graphics Menu and make all selections for one battery.
  - a. Print one Plot.
- G. View Status for all batteries.
  - a. Print Status Report for one battery.
- H. View Advice Menu and make all selections for one battery.
  - a. Print Advice Report for one battery.
- I. Quit Expert System.

5. Verify Expert System Output.
- A. Review printed Reports and Plots.